Università di Pisa
Facoltà di Scienze Matematiche Fisiche e Naturali
Dipartimento di Informatica
Corso di laurea in Tecnologie Informatiche

Tesi di laurea

Ricostruzione di superfici poligonali basata su rappresentazioni implicite

Candidato
Valentino Fiorin

. . . . . .

Relatori:
dott. Paolo Cignoni

. . . . . .

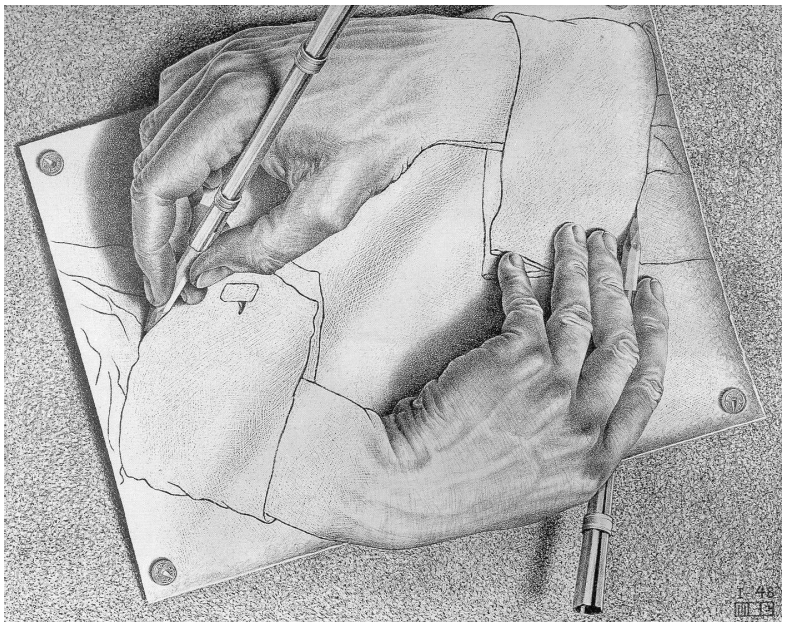
dott. Fabio Ganovelli

. . . . . .

Controrelatore:
prof. Giuseppe Attardi

. . . . . .

Anno Accademico 2004/2005



M.C. Escher. *Drawing Hands* (1948)

A mia madre e a mio padre,
per la premurosa presenza in tutti questi anni di studio.

Ai relatori ed agli amici del gruppo VCG,
per i preziosi aiuti e gli indispensabili consigli.

A Simona,
per l'incoraggiamento ed il conforto nei periodi di difficoltà.

Ricostruzione di superfici poligonali basata su rappresentazioni implicite

Valentino Fiorin

Università di Pisa
Dipartimento di Informatica

Sommario

Nell'ambito della ricostruzione di rappresentazioni digitali ottenute tramite 3D scanning, un problema importante è quello della fusione di mappe di profondità in un'unica superficie poligonale coerente e corretta. La tesi propone una tecnica innovativa per questo problema basata su una rappresentazione volumetrica implicita mantenuta su memoria secondaria e finalizzata alla gestione efficiente di grandi insiemi di range map.

Indice

Capitolo 1

Introduzione

In computer graphics con modello tridimensionale si intende una descrizione digitale di un oggetto che può essere impiegata per ottenere delle visualizzazioni dell'oggetto stesso da arbitrari punti di vista e sotto arbitrarie condizioni di illuminazione. Normalmente il modo più comune di ottenere un modello tridimensionale è quello di utilizzare appositi sistemi interattivi che, tramite l'uso di sofisticate interfacce, permettono di definire forme e superfici nello spazio. I modelli tridimensionali hanno esordito nel mercato consumer una decina di anni or sono prevalentemente all'interno del settore dell'*entertainment*. Da allora il loro utilizzo non è rimasto segregato a tale ambito, ma si è continuamente e progressivamente esteso ad innumerevoli altri settori, dall'industria all'e-commerce, dai musei virtuali alle applicazioni mediche. Il motivo di tale affermazione è da ricercarsi in un connubio di fattori che da sempre hanno caratterizzato l'informatica e che, in questo particolare ambito, sono stati di essenziale importanza: tra questi la sempre maggior potenza di calcolo dei personal computer, oggigiorno affiancata da notevoli capacità di visualizzazione grafica, la loro accessibilità ad una fascia di utenza sempre più ampia nonché la maggiore semplicità di utilizzo degli strumenti software con cui gli utenti finali interagiscono con i modelli tridimensionali. Questi progressi, uniti alle esigenze avvertite in svariati settori di avere modelli sempre più complessi e raffinati, hanno parallelamente spinto

in alcuni settori verso l'adozione delle tecniche di acquisizione tridimensionali, che allo stato attuale consentono di ottenere modelli di impressionante perfezione, fedeli copie degli originali.

La creazione di un modello tridimensionale a partire da un oggetto reale è un processo complesso che si articola in più fasi, l'ultima della quali consiste nell'individuare una qualche descrizione dell'oggetto sufficientemente pratica perché sia semplice l'interazione col modello creato. Delle possibili descrizioni, quella maggiormente impiegata, sia per la sua semplicità sia per l'estrema compatibilità con il funzionamento delle schede grafiche, è quella delle *mesh simpliciali*, in cui la superficie del modello è identificata da un insieme di triangoli connessi fra loro. Nonostante la semplicità di tale descrizione, lo step di generazione vero e proprio della mesh a partire dalla descrizione dell'oggetto reale è tutt'altro che banale e, soprattutto, non esente da errori. A causa infatti di eventuali errori nei dati, imputabili al rumore inevitabilmente presente nelle misurazioni o accumulatisi durante gli step precedenti la fase di costruzione vera e propria, il modello ricostruito potrebbe mancare di desiderabili caratteristiche che potrebbero addirittura precluderne il suo reale utilizzo: l'imperfezione da cui i modelli tridimensionali sono più comunemente affetti è l'assenza di continuità nella superficie ricostruita, che si può manifestare come triangoli adiacenti sovrapposti o sconnessi, come porzioni di superficie non two-manifold, come autointersezioni o t-junctions. Per questi e altri motivi, negli anni precedenti sono stati proposti e sviluppati svariati algoritmi il cui obiettivo era proprio l'eliminazione dalle superfici ricostruite di tali imperfezioni.

1.1 Lavoro svolto

La tesi si colloca nell'ambito della ricostruzione di superfici ed il lavoro condotto durante il suo svolgimento ha avuto per obiettivo la realizzazione di

un'applicazione capace di generare, a partire dal campionamento di un oggetto reale, il rispettivo modello tridimensionale. Poiché a questa applicazione veniva richiesta la capacità di processare input di grosse dimensioni, dell'ordine di diversi GB, abbiamo deciso di svincolare quanto più possibile l'algoritmo di ricostruzione superfici dai meccanismi che rendessero effettivamente possibile tale obiettivo. Il primo passo è stato quindi la progettazione e la successiva implementazione di un framework che si facesse carico della gestione e dell'organizzazione delle primitive di interesse, vale a dire punti, normali, colore e qualità. L'intero framework è stato costruito sopra un'octree adattivo, una struttura dati gerarchica per la gestione efficiente di dataset volumetrici, grazie al quale abbiamo potuto effettuare una sorta di indicizzazione della nuvola di punti. Il framework risultante nasconde all'esterno tutti i dettagli su come il dataset sia realmente organizzato al suo interno, mentre espone un insieme ben definito di metodi attraverso i quali è possibile accedere a sottoinsiemi contigui della nuvola di punti. Grazie a questo framework abbiamo quindi potuto concentrare i nostri sforzi sulla formulazione dell'algoritmo di estrazione della superfice. Un'attenta analisi dei metodi e delle tecniche esistenti ci ha permesso di individuare nella famiglia delle rappresentazioni implicite degli ottimi candidati per la risoluzione del nostro problema: questa classe di rappresentazioni descrive la superficie attraverso una qualche funzione, continua su tutto il volume occupato dal dataset, definita a partire dalla nuvola di punti. La tecnica da noi scelta per via delle garanzie di stabilità e di accuratezza che essa offre è definita solamente su insiemi di punti: pertanto una parte significativa del nostro lavoro ha riguardato l'individuazione di una strategia che da una descrizione a punti ci permettesse di ricavare una rappresentazione del tipo mesh simpliciale. Attraverso una reinterpretazione della rappresentazione implicita adottata siamo riusciti a definire su tutto il volume uno scalar field che combinato quindi ad un algoritmo di poligonalizzazione definito sulle celle della griglia

implicata dall'octree ha permesso di ricavare la descrizione da noi desiderata.

L'applicazione sviluppata durante questa tesi è stata capace di processare correttamente dataset comprendenti diversi milioni di punti ed ha permesso di ricavare dall'analisi del volume occupato dalla nuvola di punti dei modelli tridimensionali di sorprendente accuratezza; per di più i tempi mediamente necessari alla loro creazione si sono rivelati contrariamente anche alle nostre aspettative decisamente contenuti.

1.2 Organizzazione della tesi

Nel seguito di questo capitolo introdurremo il concetto di pipeline di acquisizione e forniremo una rapida descrizione delle sue prime fasi, quelle cioè che portano a descrivere la superficie di un modello attraverso una nuvola di punti. Nel capitolo 2 saranno invece discussi gli algoritmi di maggior rilievo e le tecniche più innovative per la generazione di un modello tridimensionale a partire da questo insieme di punti: per una maggiore chiarezza espositiva, i metodi analizzati saranno suddivisi in due grandi famiglie di algoritmi, quelli non-impliciti e quelli impliciti, a seconda della modalità con cui la superficie viene ottenuta. Nel capitolo 3 proporremo un possibile framework per la gestione efficiente di grossi insiemi di range map e per la loro interrogazione attraverso query spaziali. Su tale framework nel capitolo 4 presenteremo la tecnica da noi proposta per individuare la superficie implicata da una nuvola di punti: tale tecnica è un'estensione della rappresentazione implicita point-set che descriveremo nel capitolo 2 e mira alla conversione in mesh simpliciale della rappresentazione implicita point-set adottata come kernel del nostro algoritmo. I risultati ottenuti con questa saranno proposti ed analizzati nel capitolo 5, dove verranno discusse anche le prestazione registrate durante l'esecuzione dei test. L'ultimo capitolo lo riserviamo invece per una discussione sui possibili miglioramenti e le future estensioni.

1.3 La pipeline di acquisizione

La realizzazione di un modello tridimensionale è il risultato finale di una articolata sequenza di task che nel loro complesso vengono comunemente riassunti sotto il nome di *pipeline di acquisizione*. In questa sede non esamineremo esaustivamente lo stato dell'arte, ma ci limiteremo a fornire una panoramica sulle problematiche pertinenti le varie fasi della pipeline di acquisizione, rimandando il lettore interessato ai riferimenti in bibliografia [15, 24, 33].

Attualmente lo spettro delle tecnologie capaci di catturare una descrizione della superficie di un oggetto è piuttosto ampio e offre vari compromessi tra costi e qualità della rappresentazione fornita: i suoi estremi sono costituiti dai completi sistemi di *3D imaging*, quali i TAC scanner, capaci di fornire una accurata descrizione della superficie dell'oggetto e del loro interno, e dai molto più economici sistemi passivi come quelli, tuttora molto imprecisi, basati sulla semplice analisi di *stream video*. Non essendo essenziale ai fini della nostra trattazione un'analisi delle varie tecnologie per l'acquisizione di modelli 3D, si fornirà una descrizione di massima sul funzionamento degli scanner in uso presso il gruppo di ricerca VCG del C.N.R. di Pisa presso cui questa tesi è stata condotta, vale a dire gli scanner *a tempo di volo* e gli scanner *a luce strutturata*. Questi ultimi proiettano sull'oggetto da scandire un lama di luce generato da un laser e con un sensore, generalmente un CCD, rilevano la luce riflessa dall'oggetto: il software a corredo dello scanner computa quindi per semplice triangolazione la lista dei valori di profondità che vengono infine convertiti, sfruttando la posizione calibrata e l'orientazione sia del sensore che del fascio di luce, in coordinate tridimensionali espresse nel sistema di riferimento dello scanner. Gli scanner a tempo di volo emettono invece un piccolo impulso laser e stimano la distanza in quella precisa direzione calcolando il tempo che la luce riflessa dall'oggetto impiega per tornare allo scanner. Mentre gli scanner a luce strutturata sono maggiormente adatti

a scansioni di oggetti di dimensioni contenute, quelli a tempo di volo trovano applicazione sulle lunghe distanze, permettendo di ottenere descrizioni di interi edifici, anche se con una precisione molto inferiore.

Nonostante le intrinseche differenze fra le due tecnologie, la descrizione di un oggetto fornita da entrambe le famiglie di scanner è la stessa e prende il nome di *range map*: come detto prima si tratta di una lista di valori di profondità per un insieme di punti sull'oggetto visibili da un particolare posizione, il centro di proiezione dello scanner stesso, a partire dai quali è possibile ricavare per ogni punto le sue coordinate spaziali. In realtà ogni range map è qualcosa di più di una semplice nuvola di punti: innanzitutto la presenza di un punto in una range map implica che il punto fosse direttamente visibile dal sensore, escludendo la presenza di altre superfici occludenti, e permette inoltre di inferire in quale lato dell'oggetto il punto giaccia; essendo poi le range map organizzate come array bidimensionali, una stima della normale ad ogni punto può essere ricavata semplicemente attraverso opportuni prodotti vettoriali. Va infine fatto notare che allo stato attuale quasi tutti gli scanner sono capaci di catturare per ogni campione sia una stima dell'attendibilità del campione stesso sia dettagliate informazioni sull'aspetto della superficie in quel punto come il colore (o, più tecnicamente la riflessione spettrale). In definitiva, dal punto di vista delle successive fasi della pipeline di acquisizione, lo scanner può essere considerato una *black-box* capace di generare una nuvola di punti, ognuno dotato di normale ed eventualmente anche di colore e qualità, che approssimativamente giacciono sulla superficie dell'oggetto scandito.

La fase di acquisizione delle range map è in pratica una fase molto più complessa di quanto a prima vista possa apparire: ogni range map descrive infatti solo una porzione di superficie dell'oggetto, quella visibile dallo scanner stesso, e, per di più, questa descrizione è espressa nel sistema di riferimento dello scanner al momento dell'acquisizione. Poiché l'obiettivo finale è la

costruzione di un modello che sia consultabile da arbitrari punti di vista, è evidente la necessità di riuscire a descrivere la superficie dell'oggetto a tutto tondo: in fase di acquisizione è quindi di fondamentale importanza garantire che le range map ricoprano l'intera superficie dell'oggetto o, per lo meno, quelle porzioni di superficie a cui si è interessati.

Alla fase di acquisizione segue la fase di *registrazione* (o *allineamento*), in cui le range map vengono portate in un sistema di riferimento comune. Registrare una range map significa calcolare una trasformazione rigida che porti i punti di tale range map ad allinearsi con le porzioni di superficie che questa range map condivide con altre range map. Il problema dell'allineamento nella sua formulazione più generale, quella in cui nessuna informazione è nota a priori sulle singole range map, è di difficile soluzione e tutt'oggi l'individuazione di nuovi metodi capaci di ricavare automaticamente un allineamento iniziale attraverso metodi di *feature matching* è un attivo ambito di ricerca. In genere gli algoritmi per l'allineamento delle range map necessitano che sia loro fornito una posizione relativa anche molto approssimata delle range map, in modo da ridurre i tempi di esecuzione e da evitare allineamenti palesemente erronei. Tali allineamenti iniziali possono essere manualmente individuati da un operatore che interattivamente seleziona coppie di tre o più features condivise da range map adiacenti, oppure possono essere ricavate attraverso sistemi di *tracking* delle posizioni dello scanner (o dell'oggetto) durante la fase di acquisizione delle range map. Alla base di vari algoritmi di allineamento oggi disponibili ed estremamente diffuso per efficacia ed efficenza, è l'*iterative closest point* o ICP [9], che lavora su coppie di range map. L'algoritmo cerca innanzitutto di identificare coppie di punti probabilmente corrispondenti appartenenti all'area di sovrapposizione delle due range map; quindi, iterativamente, raffina la trasformazione rigida così da minimizzare la distanza fra le coppie di punti candidati. Ad ogni iterazione il problema di minimizzazione, efficientemente risolubile attraverso decomposizione

SVD o approcci statistici maggiormente robusti, permette di identificare una trasformazione rigida, esprimibile attraverso la composizione di una rotazione R ed una traslazione T, che riducono la distanza fra le coppie di punti, permettendo all'iterazione successiva di migliorare la scelta dei punti corrispondenti e, nel complesso, di migliorare l'allineamento. Nonostante sia stato dimostrato che l'intero processo converga ad un minimo locale, a seconda delle configurazioni iniziali l'algoritmo potrebbe anche non convergere ad un minimo globale. In letteratura sono reperibili numerosi altri algoritmi sviluppati a partire da questa idea al fine di velocizzare la convergenza o di gestire anche casi degeneri; si rimanda a [10, 30, 29] per un confronto dei metodi esistenti. In questa sede ci limitiamo ad osservare che di questo algoritmo molte varianti sono state appositamente realizzate per effettuare la registrazione di grossi dataset, in quanto, a causa dell'accumulazione degli errori, l'allineamento globale che risulterebbe da sequenziali registrazioni di coppie di range map è tutt'altro che ottimale.

La fase di registrazione si conclude quando tutte le range map sono espresse in un comune sistema di riferimento: l'oggetto è implicitamente descritto dalla nuvola di punti composta dall'insieme delle range map allineate; mancando una descrizione effettiva della superficie, ancora non si può parlare di un modello tridimensionale vero e proprio; sarà la fase di *integrazione* (anche detta di *fusione*) a costruire una superficie triangolata sfruttando le informazioni contenute nel dataset di partenza. Poiché una significativa parte del lavoro svolto in questa tesi concerne la fase di integrazione, quest'ultima sarà oggetto di ampia discussione nel capitolo 2, riservato all'analisi dei metodi esistenti per la generazione della superficie. In maniera molto sintetica per ora diciamo che durante questa fase verrà individuata la superficie dell'oggetto che costituirà una prima versione del modello tridimensionale dell'oggetto scandito.

Al fine di ottenere un valido modello tridimensionale sono spesso necessa-

rie ulteriori fasi di *postprocessing*, durante le quali il modello viene migliorato o adattato alle specifiche esigenze dell'applicazione per cui è stato creato: comuni operazioni di postprocessing includono ad esempio la semplificazione del modello ottenuto, che potrebbe essere inutilmente troppo dettagliato e la correzione di imperfezioni della superficie ricostruita. Tipiche operazioni di correzioni sono ad esempio costituite da chiusura di piccoli buchi, filtraggio di superficie affette da eccessivo rumore, o ancora eliminazione di porzioni della superficie non ritenute interessante come ad esempio i supporti utilizzati per sorreggere un oggetto durante il processo di scansione.

Capitolo 2

Algoritmi di fusione

L'obiettivo della fase di fusione è la ricostruzione di un'unica superficie triangolare continua e topologicamente corretta a partire dalle informazioni disponibili nelle range map. Queste ultime, che alla conclusione della precedente fase di allineamento sono espresse in un unico sistema di riferimento, costituiscono l'input a partire dal quale gli algoritmi di fusione generano il modello tridimensionale dell'oggetto scandito. La descrizione della pipeline di acquisizione, delineata nel capitolo precedente, verrà ora conclusa con l'analisi della fase di fusione. Nel seguito del capitolo verranno analizzati gli algoritmi di fusione di maggior importanza, fornendo per ognuno di essi una descrizione di massima sul procedimento adottato per ricostruire un valido modello tridimensionale a partire dall'insieme delle range map. Per chiarezza di esposizione nella trattazione che segue saranno introdotte due famiglie di algoritmi così da fare una prima grossa classificazione tra i metodi che impiegano una descrizione implicita della superficie e quelli che invece adottano una descrizione più tradizionale.

2.1 Rappresentazioni non implicite

2.1.1 Mesh Zippering

Fra i vari algoritmi che si approfondiranno, il mesh zippering [35] è stato di certo il primo a fornire una soluzione abbastanza robusta e generale al problema della fusione ed è inoltre quello che forse più degli altri sfrutta le informazioni topologiche presenti nelle range map per costruire la triangolarizzazione della superficie finale. L'idea alla sua base è quella di combinare progressivamente alla mesh fino a quel momento ricostruita le range map con cui questa condivida una porzione di superficie: i bordi delle range map coinvolte nella fusione vengono prima "levigati" ed i triangoli dei loro bordi vengono quindi fatti combaciare attraverso la condivisione dei vertici di bordo. Quando tutte le range map sono state combinate, la posizione di ogni vertice nella mesh finale viene raffinata pesando il contributo proveniente da ogni range map contenente quel punto.

La fusione di due range map adiacenti è preceduta da una fase di preprocessing dei loro bordi, durante la quale vengono progressivamente rimosse porzioni ridondanti di superficie, quelle cioè in cui le due range map si sovrappongono, e che si conclude solamente quando le due range map risultano non più modificabili. Ogni triangolo appartenente alla zona di sovrapposizione di range map adiacenti viene marcato come ridondante per poi essere rimosso a seconda della posizione occupata dalla sua proiezione sull'altra range map. Siano v_1, v_2 e v_3 i vertici del triangolo T appartenente alla range map A, e siano v'_1, v'_2 e v'_3 i punti loro più vicini sulla range map B: il triangolo T viene marcato come ridondante se la distanza fra ogni suo vertice v_i e la corrispondente proiezione v'_i è minore di un valore di tolleranza d e se inoltre il punto v'_i appartiene al bordo di B. Se le range map includono inoltre una misura di confidenza sulla posizione di ogni vertice, la procedura

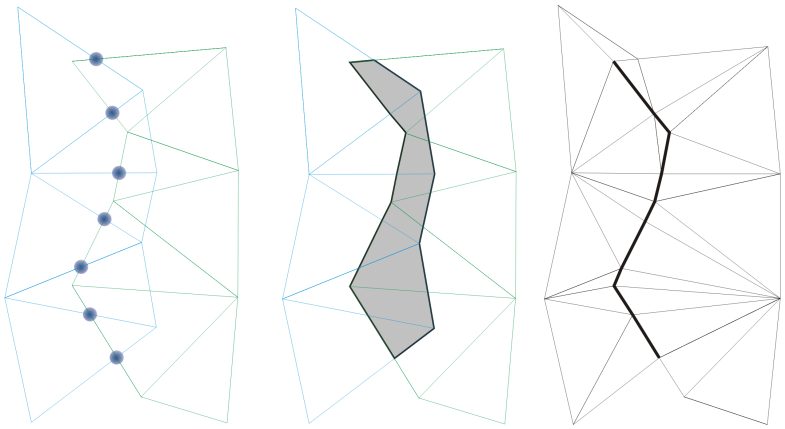


Figura 2.1: *Clipping di due mesh lungo i bordi. A sinistra, i cerchi in blu evidenziano le intersezioni fra gli edge di bordo della mesh di sinistra e quella di destra. Al centro, una porzione dei triangoli della mesh di sinistra (evidenziato in grigio) deve essere rimosso. A destra, le due mesh sono aggiornate in modo tale da incorporare i punti di intersezione.*

di eliminazione può essere modificata così da sfruttare questa informazione aggiuntiva: in questo caso, un triangolo viene marcato e poi rimosso se, oltre a verificare le condizioni di cui sopra, almeno due dei suoi vertici hanno misura di confidenza minore delle loro rispettive proiezioni. I buchi che a causa di questa procedura possono crearsi nella zona di unione di due range map sono di dimensioni ridotte e possono essere agilmente coperti durante una fase conclusiva di post-processing.

Il processo di eliminazione delle superfici ridondanti lascia lungo i bordi delle range map dei triangoli che leggermente si sovrappongono: durante la fase di *clipping*, la topologia ai bordi delle range map viene aggiustata così da poterle fondere in un'unica superficie smooth come illustrato in figura 2.1. Per chiarezza di esposizione, il processo di clipping verrà illustrato sotto l'assunzione che le due range map coinvolte giacciano sullo stesso piano; l'estensione al caso generico risulterà immediato e banale. Affinché i bordi

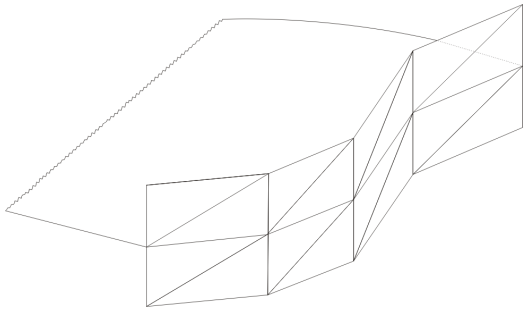


Figura 2.2: *Bordo per il clipping in tre dimensioni.*

di due range map A e B combacino perfettamente, è necessario aggiungere un nuovo vertice ad ogni intersezione fra gli spigoli dei triangoli in A ed il bordo di B; l'unione di Q, l'insieme dei vertici così aggiunti, con l'insieme dei vertici di bordo in B costituiscono una frontiera comune che dovrà essere condivisa dai triangoli ai bordi delle due range map. Ogni triangolo in B viene scisso una volta per ogni nuovo vertice inserito; dei triangoli in A solamente una parte verrà utilizzata nella ricostruzione della superficie finale, in quanto la porzione sovrapposta al bordo di B non viene utilizzata: i vertici appartenenti alla porzione di superficie non sovrapposta a B vengono infine passati ad una routine che provvede ad identificare l'insieme di triangoli necessari a ricoprire tale area. Tale procedimento può essere esteso facilmente ad uno spazio tridimensionale, in cui l'unica difficoltà in più è costituita dal possibile disallineamento fra le due superfici. Questo caso viene gestito immaginando che il bordo della range map B sia delimitato da una barriera di triangoli localmente perpendicolare alla superficie in B (vedi figura 2.2): il punto di intersezione della range map A con B può essere facilmente individuato calcolando dove A intersechi la barriera e quindi proiettando tale punto sull'edge di bordo di B.

Come il lettore più attento avrà certamente notato, la fase di clipping ha

la tendenza a generare insiemi di triangoli molto piccoli o sottili, che possono comunque essere facilmente rilevati e rimossi attraverso un algoritmo di *vertex deletion*: se il rapporto tra l'area ed il perimetro di un triangolo è inferiore ad un valore soglia specificato dall'utente, vengono eliminati uno dei suoi vertici e tutti i triangoli che condividono questo vertice; il buco che si viene a creare può essere tappato generando una nuova triangolarizzazione.

2.1.2 Crust

La superficie dell'oggetto viene ricostruita da quest'algoritmo utilizzando la triangolarizzazione di Delaunay di un opportuno sottoinsieme dei vertici appartenenti al diagramma tridimensionale di Voronoi per i punti nel dataset. Per conformità alla nomenclatura utilizzata nell'articolo originale [3], in questa sede chiameremo *crust* la superficie ricostruita. Poiché quest'ultima viene estratta dalla triangolarizzazione di Delaunay, i suoi vertici coincidono con i punti presenti nel dataset di input: il crust di conseguenza interpola la superficie dell'oggetto anziché approssimarla. Sotto stringenti vincoli sulla densità del dataset di partenza è stata dimostrata la correttezza dell'algoritmo, sia nel senso di una fedele ricostruzione topologica, sia di convergenza alla superficie reale per campionamenti di densità crescenti; in pratica l'algoritmo ha generato risultati soddisfacenti anche per dataset con densità di campionamento inferiore al limite teorico. I vincoli circa la qualità del campionamento vengono formalmente espressi in funzione del *medial axis*; riprenderemo fra breve quest'aspetto, dopo aver introdotto alcune nozioni geometriche utili per la completa comprensione dell'algoritmo.

Dato un insieme finito $\mathscr{C}$ di punti campionati in $\mathbb{R}^d$, viene detta *cella di Voronoi* di un punto campione quel sottospazio di $\mathbb{R}^d$ a lui più vicino rispetto ad ogni altro campione in $\mathscr{C}$; ogni cella di Voronoi è un politopo convesso i cui vertici sono i cosiddetti *vertici di Voronoi*. La decomposizione

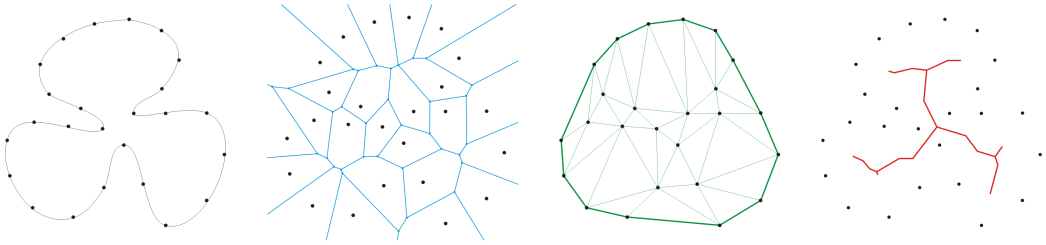


Figura 2.3: *Diagramma di Voronoi, simplesso di Delaunay e medial axis. A sinistra l'insieme di punti $\mathscr{C}$ campionato sulla figura rappresentata dalla linea continua. In (b), il diagramma di Voronoi di $\mathscr{C}$: i punti in blu sono i vertici $\mathscr{V}$ delle celle di Voronoi. In (c), il simplesso di Delaunay. A destra infine la linea rossa indica il medial axis di $\mathscr{C}$.*

di $\mathbb{R}^d$ indotta dalle celle di Voronoi prende il nome di *diagramma di Voronoi*. In figura 2.3 è riportato un esempio per il caso bidimensionale: in (a) un campionamento $\mathscr{C}$ a partire dalla sagoma rappresentata dalla linea continua ed in (b) il rispettivo diagramma di Voronoi. Quando $\mathscr{C}$ è non degenere, ogni vertice di Voronoi è equidistante ad esattamente $d+1$ punti di $\mathscr{C}$: tali $d+1$ punti sono i vertici del *simplesso di Delaunay*, duale rispetto ai vertici di Voronoi; ogni faccia del simplesso di Delaunay risulta quindi circoscritta in una sfera all'interno della quale non cadono altri punti di $\mathscr{C}$. L'insieme dei simplessi di Delaunay costituisce la *triangolarizzazione di Delaunay* di $\mathscr{C}$ (figura 2.3 (c)). Infine, il *medial axis* di una superficie $(d-1)$-dimensionale in $\mathbb{R}^d$ è l'insieme dei punti con più di un punto a distanza minima dalla superficie (figura 2.3(d)): il medial axis è perciò un'estensione alle superfici continue del diagramma di Voronoi, nel senso che il diagramma di Voronoi di un insieme $\mathscr{C}$ può essere definito come l'insieme dei punti con più di un punto a distanza minima in $\mathscr{C}$.

Avendo introdotto i concetti indispensabili per l'analisi dell'algoritmo, è

possibile ora fornire la definizione di buon campionamento che si era prima lasciata in sospeso. Il vincolo che l'algoritmo impone al campionamento concerne la sua densità, che deve risultare *almeno* inversamente proporzionale alla distanza dal medial axis; più precisamente si definisce un campionamento, costituito da un insieme di punti $\mathscr{C}$, un *r-sample* di una superficie $\mathscr{S}$ se la distanza euclidea fra ogni punto della superficie $\mathscr{S}$ ed il più vicino campione $\boldsymbol{c} \in \mathscr{C}$ è al più r volte la distanza fra $\boldsymbol{c}$ ed il più vicino punto del medial axis di $\mathscr{S}$. Questa definizione cattura la nozione intuitiva che, al fine di una corretta ricostruzione della superficie, la densità del campionamento è un parametro che può variare localmente, in quanto aree ricche di features necessitano di una maggiore densità, mentre in quelle più regolari i campioni possono anche essere più sparsi. Infatti in tale definizione sono implicitamente chiamate in causa sia la curvatura di una superficie, in quanto il medial axis è tanto più vicino alla superficie quanto più la curvatura è maggiore, sia la prossimità di altre porzioni di superficie.

Per facilitare la comprensione dell'algoritmo, se ne descriverà il funzionamento nel più semplice caso bidimensionale, rimandando a dopo la spiegazione delle modifiche necessarie per l'estensione al caso tridimensionale. Sia $\mathscr{C}$ l'r-sample, con r sufficientemente piccolo ($r \leqslant 0.25$), campionato a partire da una curva in $\mathbb{R}^2$. Dopo aver costruito il diagramma di Voronoi di $\mathscr{C}$, si definisce un nuovo insieme $\mathscr{Q} = \mathscr{C} \cup \mathscr{V}$ comprendente oltre ai campioni iniziali in $\mathscr{C}$ anche l'insieme $\mathscr{V}$ dei vertici di Voronoi. Il crust è un sottoinsieme dalla triangolarizzazione di Delaunay di $\mathscr{Q}$ individuato scegliendo solamente quegli archi e che connettono punti di $\mathscr{C}$ adiacenti e il cui cerchio circoscritto non contenga, al di fuori dei due estremi di e, né punti campione in $\mathscr{C}$, né vertici di Voronoi in $\mathscr{V}$ (vedi figura 2.4). La tecnica di utilizzare anche i vertici di Voronoi prende il nome di *Voronoi filtering* e garantisce che nel crust siano contenuti solamente quel sottoinsieme di archi della triangolarizzazione di Delaunay necessaria a descrivere la curva da ricostruire. Infatti i vertici di

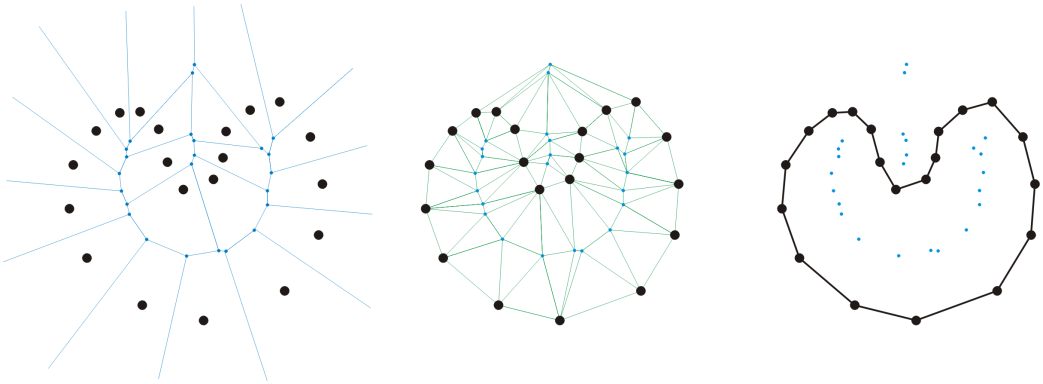


Figura 2.4: *Il crust nel caso bidimensionale. A sinistra, il diagramma di Voronoi di un insieme di punti $\mathscr{C}$ campionati su una curva: come è possibile notare, i vertici di Voronoi $\mathscr{V}$ approssimano il medial axis della curva. Al centro, la triangolarizzazione di Delaunay dell'insieme di punti $\mathscr{C} \cup \mathscr{V}$. A destra, il crust della nuvola di punti $\mathscr{C}$.*

Voronoi costituiscono, almeno in $\mathbb{R}^2$, una buona approssimazione del medial axis: di conseguenza, la tecnica di scartare quegli edge della triangolarizzazione di Delaunay il cui cerchio circoscritto contiene vertici in $\mathscr{V}$ garantisce che il crust sia costituito solamente da edge fra coppie adiacenti di vertici, per i quali il medial axis risulta per definizione lontano.

L'estensione dell'algoritmo al caso tridimensionale non è immediato dato che la tecnica del Voronoi filtering non è più sufficiente ad individuare le componenti superflue prodotte dalla triangolarizzazione di Delaunay: infatti in $\mathbb{R}^3$ i vertici di Voronoi non costituiscono più una rappresentazione del medial axis ma possono occupare posizioni arbitrariamente vicine alla superficie. Fortunatamente però un sottoinsieme dei vertici di Voronoi costituiscono ancora una buona approssimazione del medial axis. Si consideri la cella di Voronoi $V_{\boldsymbol{c}}$ associata ad un campione $\boldsymbol{c} \in \mathscr{C}$ illustrata in figura 2.5. Intorno a $\boldsymbol{c}$ giacciono altri punti campione e $V_{\boldsymbol{c}}$ risulta pertanto limitata da piani di bisezione, grosso modo perpendicolari alla superficie $\mathscr{S}$, che

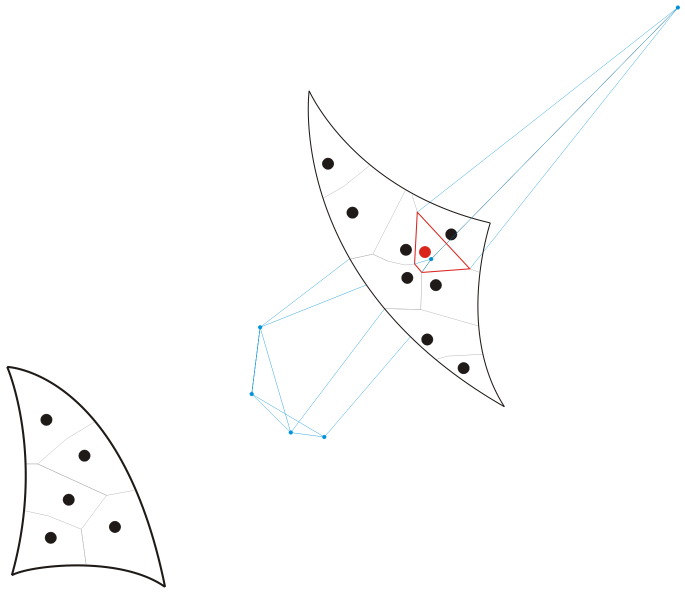


Figura 2.5: *In uno spazio tridimensionale, non tutti i vertici di Voronoi $\mathscr{V}$ sono direttamente utilizzabili per approssimare il medial axis. Si consideri il campione in rosso: gli edge ed i vertici della sua cella tridimensionale di Voronoi sono evidenziati in blu: uno di tali vertici giace in prossimità della superficie, in una posizione equidistante dagli altri quattro campioni al centro della porzione di superficie; gli altri vertici invece approssimano il medial axis, in prossimità del centro di curvatura da un lato, e dall'altro a metà strada rispetto ad un altra porzione di superficie.*

separano $\boldsymbol{c}$ dai suoi vicini: pertanto la cella $V_{\boldsymbol{c}}$ risulta sottile ed allungata e quasi perpendicolare alla superficie in $\mathscr{S}$, estendendosi verso il medial axis; in prossimità di quest'ultimo, la cella $V_{\boldsymbol{c}}$ viene tagliata dato che altri campioni in $\mathscr{C}$ diventano più vicini di $\boldsymbol{c}$, garantendo quindi che alcuni vertici della cella $V_{\boldsymbol{c}}$ giacciano nei pressi del medial axis. Le modifiche da apportare allo step di Voronoi filtering riguardano solamente la scelta dei vertici di Voronoi da utilizzare: per la precisione, anziché utilizzare tutti i vertici come nel caso bidimensionale, lo step di Voronoi filtering sfrutta di ogni cella $V_{\boldsymbol{c}}$ solamente i suoi due vertici che risultano più lontani dal punto campione $\boldsymbol{c}$ e che giacciono su semispazi opposti della superficie: nel seguito chiameremo questi due vertici i *poli* di $\boldsymbol{c}$ e li indicheremo con $\boldsymbol{p}_{\boldsymbol{c}}^{+}$ e $\boldsymbol{p}_{\boldsymbol{c}}^{-}$. Il punto di partenza dell'algoritmo rimane sempre la costruzione del diagramma di Voronoi del campionamento $\mathscr{C}$. Per ogni punto $\boldsymbol{c}$ è necessario quindi individuare i suoi due poli: $\boldsymbol{p}_{\boldsymbol{c}}^{+}$ è semplicemente il vertice della cella di Voronoi $V_{\boldsymbol{c}}$ a maggior distanza da $\boldsymbol{c}$, mentre l'altro polo $\boldsymbol{p}_{\boldsymbol{c}}^{-}$ viene calcolato come il vertice della cella di Voronoi $V_{\boldsymbol{c}}$ a maggior distanza da $\boldsymbol{c}$ ma con proiezione negativa su $\vec{\boldsymbol{n}}^{+}$; se $\boldsymbol{c}$ giace sul *convex hull* questo vettore viene calcolato come media delle normali ai triangoli adiacenti, altrimenti come $\boldsymbol{s} \cdot \boldsymbol{p}_{\boldsymbol{c}}^{+}$. Dopo aver individuato l'insieme $\mathscr{P}$ dei poli di ogni punto $\boldsymbol{c}$, si calcola la triangolarizzazione di Delaunay dell'insieme $\mathscr{C} \cup \mathscr{P}$. Il crust è quindi ottenuto semplicemente scartando da questa poligonalizzazione quei triangoli in cui almeno un vertice è un polo appartenente all'insieme $\mathscr{P}$.

2.1.3 Ball pivoting

L'algoritmo segue l'approccio *region growing* utilizzando come operazione di base l'omonima ball pivoting [8]: una ρ-ball, cioè una sfera avente raggio fisso ρ, è inizialmente messa in contatto con tre punti del dataset, che definiscono il primo triangolo (*seed triangle*) della superficie. I tre suoi spigoli inizializzano

una coda di edge che rappresentano la frontiera attiva $\mathscr{F}$; iterativamente da questa coda viene estratto un edge ed intorno a questo viene fatta ruotare la sfera finché non entra in collisione con un nuovo punto. Questo nuovo punto, insieme ai due estremi dell'edge, definiscono un nuovo triangolo della superficie: la regione attiva viene aggiornata ed il processo continua finché non vengono processate tutta le porzioni connesse in questo modo raggiungibili.

Il ball pivoting risulta strettamente connesso con le α-shapes, uno dei primi validi algoritmi di integrazione, molte delle cui proprietà possono essere dimostrate anche per quest'algoritmo. Le α-shapes [14] rappresentano un'intera famiglia di superfici, che vengono generate in funzione del parametro α. La costruzione della superficie per un prefissato valore di α si appoggia alla poligonalizzazione di Delaunay dell'insieme di punti in input $\mathscr{C}$: l'α-shape viene quindi ricavata da tale poligonalizzazione scegliendo solamente quei simplessi σ che risultino inscrivibili in una sfera di raggio $\rho \leqslant \sqrt{\alpha}$ che non contenga altri punti di $\mathscr{C}$ diversi dai vertici di σ.

Sebbene diretto erede delle α-shapes, il ball pivoting riuscì a superare molte delle loro limitazioni: le α-shapes infatti risultano estremamente sensibili al rumore presente nelle range map e, soprattutto, la loro applicazione è limitata a dataset di ridotte dimensioni in quanto la superficie ricostruita è un sottoinsieme della triangolarizzazione di Delaunay, problema non facilmente risolubile per insiemi di milioni di punti.

Due semplici vincoli sono sufficienti al ball pivoting per garantire una veloce ed accurata ricostruzione della superficie: il primo concerne il raggio di curvatura della superficie, che deve essere non minore di ρ, cosicché la ρ-ball possa passare attraverso feature concave evitando multipli contatti con la superficie; la seconda riguarda invece la densità di campionamento, che deve risultare sufficientemente densa perché la ρ-ball possa camminare sul manifold senza lasciare buchi. Se i dati soddisfano tali condizioni, un seed triangle per ogni componente connessa è sufficiente per la costruzione

di un manifold. Un valido seed triangle può essere individuato osservando semplici regole: innanzitutto uno dei suoi vertici deve essere scelto tra i punti non ancora utilizzati nella triangolarizzazione: sia v tale punto; gli altri due vertici, v_a e v_b, sono coppie di punti scelti nell'intorno di v e presi in ordine decrescente di distanza da v. Perché questi tre punti definiscano un valido seed triangle, il vettore perpendicolare al triangolo, che deve puntare verso l'esterno della superficie, deve essere consistente con le normali ai vertici ed inoltre la ρ-ball, definita cosicché i tre vertici giacciano sulla sua superficie, deve essere centrata nel semipiano esterno e non deve contenere altri punti. A partire da un valido seed triangle, l'algoritmo costruisce la componente connessa contenente tale triangolo.

L'operazione base con cui la superficie viene progressivamente costruita è denominata *ball pivoting* ed è illustrata in figura 2.6. Si consideri un triangolo T di vertici v_0, v_1 e v_2 ed una ρ-ball che sia circoscritta dai vertici di T, e si assuma, senza perdità di generalità, che lo spigolo $e_{(i,j)} = e_{(1,2)}$ sia il *pivoting edge*. Nella sua posizione iniziale la sfera è centrata in un punto $\boldsymbol{c}$ e, o perché T è un seed triangle o perché T è stato individuato durante un operazione di pivoting eseguita in precedenza, al suo interno non sono contenuti altri punti del dataset al di fuori dei vertici di T. Poiché l'operatore di pivoting può essere visto come una rotazione avente per asse il pivoting edge, durante la sua applicazione la ρ-ball è vincolata a stare in contatto con lo spigolo $e_{(i,j)}$, ed il suo centro $\boldsymbol{c}$ descrive una traiettoria circolare γ che giace su un piano che è perpendicolare al pivoting edge e che passa per il punto di mezzo $\boldsymbol{m} = \frac{v_i+v_j}{2}$ di quest'ultimo. Durante tale rotazione la ρ-ball potrebbe entrare in collisione con qualsiasi altro punto $\boldsymbol{x}_h$ del dataset a distanza non maggiore di 2ρ da $\boldsymbol{m}$. Se non viene intercettato alcun punto, allora lo spigolo $e_{(i,j)}$ viene classificato come edge di bordo e l'algoritmo riprende la ricostruzione della superficie dal primo elemento nella frontiera attiva $\mathscr{F}$. Altrimenti, il triangolo T' di vertici v_0, v_1 e v_h è un nuovo triangolo valido e la sfera nella

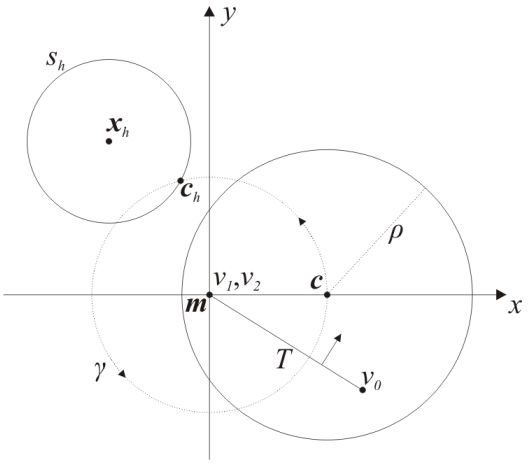


Figura 2.6: *L'operazione di ball pivoting. Il pivoting edge giace sull'asse z, che risulta perpendicolare alla pagina ed orientato verso il lettore; il suo centro è il punto* $\boldsymbol{m}$*, che coincide con l'origine del sistema di riferimento.* s_h *è invece l'intersezione con il piano* $z = 0$ *della sfera di raggio* ρ *centrata in* $\boldsymbol{x}_h$*.*

sua posizione finale individua un nuovo valido punto di partenza per una successiva operazione di pivoting, dal momento che essa non contiene altri punti al di fuori dei tre vertici.

Fra tutti i punti $\boldsymbol{x}'_h$ del dataset intercettati dalla ρ-ball durante la rotazione intorno a $\boldsymbol{m}$, il vertice v_h del nuovo triangolo T' viene individuato in base all'ordine di collisione. Più precisamente, per ogni punto $\boldsymbol{x}'_h$ viene determinato il centro $\boldsymbol{c}'_h$ della sfera passante per v_i, v_j e $\boldsymbol{x}'_h$, sempre che sia possibile definirla; i centri $\boldsymbol{c}'_h$ giacciono tutti sulla traiettoria circolare γ intorno ad $\boldsymbol{m}$ e possono essere calcolati come il punto di intersezione fra la circonferenza γ e la ρ-ball centrata in $\boldsymbol{x}'_h$. Il primo centro $\boldsymbol{c}'_h$ ad essere intercettato dalla traiettoria γ diventa il terzo vertice v_h del nuovo triangolo T'.

I punti che vengono intersecati dalle ρ-ball durante le operazioni di pivoting vengono utilizzati per l'inserimento di nuovi triangoli nella mesh ed il

successivo aggiornamento della frontiera attiva $\mathscr{F}$. Nel più semplice dei casi il punto v_k individuato dall'operazione di pivoting intorno all'edge $e_{(i,j)}$ non è stato ancora utilizzato nella triangolarizzazione della superficie: attraverso l'operazione di `join`, il triangolo di vertici v_i, v_j e v_k è aggiunto alla mesh e la frontiera attiva è localmente modificata rimuovendo lo spigolo $e_{(i,j)}$ ed inserendo i due nuovi edge $e_{(i,k)}$ e $e_{(k,j)}$. Se invece il vertice v_k fa già parte della superficie, operazioni diverse vengono eseguite a seconda che sia un vertice interno alla mesh o che ne sia vertice di bordo. Nel primo caso, il triangolo corrispondente non deve essere generato, perché si verrebbe a creare un vertice non manifold, e l'edge $e_{(i,j)}$ viene marcato come edge di bordo; nell'altro caso, invece, dopo aver verificato che l'orientazione sia consistente, si esegue una operazione di `join` creando il nuovo triangolo v_i, v_k e v_j; eventuali coppie di spigoli coincidenti ma con orientazioni diverse che si possono creare in questa circostanze vengono poi rimosse attraverso l'operazione di `glue`. Questa routine elimina dalla frontiera attiva quelle coppie di spigoli eventi stessi estremi ma diversa orientazione. Se, per esempio, un'operazione di `join` aggiunge alla frontiera lo spigolo $e_{(i,j)}$ quando già questa contiene lo spigolo opposto $e_{(j,i)}$, allora un'operazione di `glue` rimuove entrambi i nuovi spigoli ed aggiorna opportunamente la frontiera così da lasciarla in uno stato consistente.

2.2 Rappresentazioni implicite

Gli algoritmi fin ad ora descritti sono tutti accomunati dalla caratteristica di impiegare più o meno direttamente i punti del dataset come vertici del modello ricostruito, e di interpolarne la superficie attraverso una loro opportuna triangolarizzazione. Gli algoritmi appartenenti alla famiglia di rappresentazioni implicite utilizzano invece i punti nel dataset per generare una qualche funzione f continua in $\mathbb{R}^3$ e definita sull'intero volume occupato dal data-

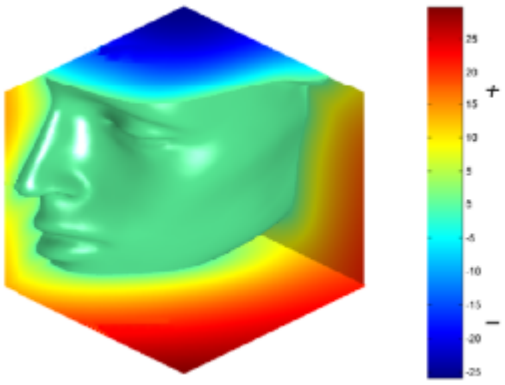


Figura 2.7: *Visualizzazione della funzione distanza segnata: la funzione assume valori positivi all'esterno della superficie e valori negativi al suo interno; il suo insieme zero $Z(f)$ implicitamente rappresenta la superficie.*

set, attraverso la cui valutazione generare infine la superficie *implicata* dalla nuvola di punti: questa funzione assume generalmente valore positivo all'esterno della superficie e valore negativo al suo interno, e l'insieme $Z(f)$ dei punti dove essa si annulla, detto *insieme zero* di f, definisce *implicitamente* una superficie che in modo smooth interpola i punti nel dataset (vedi figura 2.7). La proprietà di definire una superficie approssimante, sebbene possa apparire come un difetto, in realtà costituisce un punto di forza di questa famiglia di algoritmi, dato che il dataset contiene quasi certamente delle imperfezioni derivanti dalle precedenti fasi di campionamento e allineamento. Questa nuova tecnica, oltre ad essere più robusta di fronte a tali imperfezioni, definisce anche una base comune alla soluzione di problemi onnipresenti in computer graphics: tra questi la riparazione di mesh, il *remeshing* di modelli esistenti, sia per una loro semplificazione (*downsampling*) sia per l'estrapolazione di maggiori informazioni (*upsampling*), l'efficiente implementazione di test sulla superficie, per lo più impiegati per *ray tracing* e *collision detection*,

e, non ultimo, la creazione di accurati modelli poligonali.

2.2.1 Algoritmi zero-set

Il primo algoritmo di rappresentazione implicita che descriveremo [16] nasce con l'obiettivo di fornire una soluzione al problema della fusione nella sua formulazione più generale, quella cioè in cui nessuna informazione è nota a priori circa la superficie da ricostruire: la topologia, la presenza di bordi e la geometria del modello vengono tutte dedotte automaticamente a partire dalla nuvola di punti. Su quest'ultima l'algoritmo non impone neanche particolari vincoli, ma richiede siano note solamente alcune informazioni di massima circa il processo di campionamento in termini di densità e di limite alla magnitudo del rumore, al fine di garantire una corretta ricostruzione della superficie.

La prima fase dell'algoritmo è mirata alla definizione di una funzione $f : D \to \mathbb{R}$ definita in un intorno $D \subset \mathbb{R}^3$ del dataset che sia una valida stima della distanza segnata dalla superficie. Quest'ultima, non essendo ancora definita in questa fase dell'algoritmo, viene approssimata associando ad ogni punto del dataset un piano orientato: tali piani sono localmente tangenti alla superficie e ne costituiscono una ragionevole approssimazione lineare locale. Sfruttando questi piani, la distanza di un punto $\boldsymbol{p} \in \mathbb{R}^3$ dalla superficie viene approssimata come la distanza fra $\boldsymbol{p}$ ed il piano a lui più vicino.

Il piano tangente $Tp(\boldsymbol{x}_i)$ associato ad un campione $\boldsymbol{x}_i$ è rappresentato con una coppia $(\boldsymbol{c}_i, \vec{\boldsymbol{n}}_i)$, che indicano rispettivamente il centro del piano e la sua normale; il calcolo di ogni piano tangente $Tp(\boldsymbol{x}_i)$ passa attraverso l'insieme $k\text{-}Nbhd(\boldsymbol{x}_i)$ dei k punti più vicini al campione $\boldsymbol{x}_i$. Il valore k può essere nel caso più semplice un parametro globale specificato dall'utente oppure può essere gestito automaticamente dall'algoritmo che lo modifica così da adattarlo alle caratteristiche locali della superficie e/o del dataset. Il piano

$Tp(\boldsymbol{x}_i)$ viene costruito così da risultare la miglior approssimazione, nel senso dei minimi quadrati, dell'insieme k-neighborhood: infatti la sua componente $\boldsymbol{c}_i$ viene definita come il centro di massa dei punti in k-neighborhood mentre la normale $\vec{\boldsymbol{n}}_i$ al piano è calcolata analizzando la matrice di covarianza associata all'insieme k-neighborhood. Questa è una matrice 3×3 definita come:

$$CV = \sum_{\boldsymbol{y} \in Nbhd(\boldsymbol{x}_i)} (\boldsymbol{y} - \boldsymbol{c}_i) \times (\boldsymbol{y} - \boldsymbol{c}_i), \tag{2.1}$$

in cui l'operatore $\times$ denota il prodotto vettoriale standard. Se $\lambda_i^1 \geqslant \lambda_i^2 \geqslant \lambda_i^3$ denotano i tre autovalori di CV associati rispettivamente ai tre autovettori $\vec{\boldsymbol{v}}_i^1$, $\vec{\boldsymbol{v}}_i^2$ e $\vec{\boldsymbol{v}}_i^3$, allora la normale $\vec{\boldsymbol{n}}_i$ del piano $Tp(\boldsymbol{x}_i)$ è impostata a $\vec{\boldsymbol{v}}_i^3$ oppure a $-\vec{\boldsymbol{v}}_i^3$ a seconda dell'orientazione dei piani adiacenti.

La consistente orientazione dei piani è una condizione necessaria a garantire che la superficie ricostruita sia manifold e costituisce dell'intero algoritmo il punto più delicato e complesso. Intuitivamente, se $\boldsymbol{x}_i$ e $\boldsymbol{x}_j$ sono due punti del dataset e la superficie da ricostruire è in questo punto sufficientemente smooth, allora i loro piani tangenti $Tp(\boldsymbol{x}_i) = (\boldsymbol{c}_i, \vec{\boldsymbol{n}}_i)$ e $Tp(\boldsymbol{x}_j) = (\boldsymbol{c}_j, \vec{\boldsymbol{n}}_j)$ sono quasi paralleli, vale a dire $\vec{\boldsymbol{n}}_i \cdot \vec{\boldsymbol{n}}_j \approx \pm 1$; più precisamente, se i due piani sono consistentemente orientati, allora $\vec{\boldsymbol{n}}_i \cdot \vec{\boldsymbol{n}}_j \approx +1$, altrimenti $\vec{\boldsymbol{n}}_i \cdot \vec{\boldsymbol{n}}_j \approx -1$ e una delle due normali, o $\vec{\boldsymbol{n}}_i$ o $\vec{\boldsymbol{n}}_j$, è orientata nel verso sbagliato. Individuare un'orientazione globalmente consistente è un problema di non immediata soluzione in quanto implica imporre quest'ultima proprietà ad ogni coppia di punti "sufficientemente vicini". In realtà questo problema può essere riformulato come problema combinatorio sui grafi, ed essere quindi risolto utilizzando algoritmi standard in questo ambito.

Si costruisce un grafo il cui insieme di nodi è l'insieme dei centri $\boldsymbol{c}_i$ dei piani tangenti ed il cui insieme degli archi è costituito da tutti e soli gli edge $e_{i,j} = (\boldsymbol{c}_i, \boldsymbol{c}_j)$ per cui o $\boldsymbol{c}_i \in Nbhd(\boldsymbol{n}_j)$ o $\boldsymbol{c}_j \in Nbhd(\boldsymbol{n}_i)$; in questo grafo quindi un arco connette due nodi solamente se i piani corrispondenti

risultano geometricamente vicini, catturando in questo modo la nozione di vicinanza a cui prima si era accennato. Ad ogni arco $e_{(i,j)}$ nel grafo viene quindi associato il costo $1-\|\vec{\boldsymbol{n}}_i \cdot \vec{\boldsymbol{n}}_j\|$ che, oltre ad essere non negativo, è tanto minore quanto più i due piani (non orientati) risultano paralleli. Una volta scelta un'orientazione corretta per un qualche piano, l'ordine di propagazione lungo il *minimal spanning tree* dell'albero radicato in tal nodo garantisce una orientazione globalmente consistente delle normali dal momento che l'MST tende a visitare prima gli archi di peso minore, i quali, per come il problema è stato formulato, corrispondono alle porzioni di superficie con bassa curvatura.

I piani tangenti sono a questo punto consistentemente orientati e la distanza segnata $f(\boldsymbol{p})$ di un punto $\boldsymbol{p} \in \mathbb{R}^3$ dalla superficie può essere approssimata con la distanza segnata di $\boldsymbol{p}$ dal piano il cui centro $\boldsymbol{c}_i$ è più vicino a $\boldsymbol{p}$, vale a dire $f(\boldsymbol{p}) = dist_i(\boldsymbol{p}) = (\boldsymbol{p} - \boldsymbol{c}_i) \cdot \vec{\boldsymbol{n}}_i$. La superficie dell'oggetto può quindi essere agevolmente ricostruita sfruttando un qualche algoritmo, quale ad esempio Marching Cubes [22], capace di estrarre dall'isosuperficie di valore zero della funzione f una rappresentazione simpliciale continua.

2.2.2 Radial Basis Function

I primi tentativi verso la rappresentazione implicita di superfici esprimevano la funzione f attraverso le *radial basis function* per via delle interessanti proprietà che queste funzioni esibiscono; tuttavia la loro applicazione rimase circosritta a problemi di dimensione molto limitate a causa della complessità computazionale richiesta. L'algoritmo che verrà ora descritto [11, 12] fu il primo a consentire l'applicazione di questa nuova base teorica a problemi reali grazie all'adozione di un approccio greedy che iterativamente modifica la rappresentazione della superficie fino al raggiungimento di un prefissato livello d'accuratezza.

Il punto di partenza è, come per il metodo descritto in precedenza, la

Figura 2.8: *Ricostruzione di una mano a partire da una nuvola di punti: in verde i punti sulla superficie, in rosso i punti off-surface esterni ed in blu quelli interni. Senza la convalida dei punti off-surface, la superficie ricostruita può presentare delle autointersezioni (figura al centro), che possono essere rimosse semplicemente imponendo che il punto più vicino ad ogni punto off-surface sia il punto on-surface attraversi cui è stato generato (figura a destra).*

definizione di una funzione f attraverso cui descrivere implicitamente la superficie $\mathscr{S}$ dell'oggetto; in questo caso vogliamo che l'insieme zero $Z(f)$ di tale funzione contenga almeno i punti $\boldsymbol{x}_i$ nel dataset, e pertanto f deve necessariamente rispettare il vincolo:

$$f(\boldsymbol{x}_i) = 0 \qquad i = 1, \ldots, n.$$

Per evitare la soluzione banale in cui f è costantemente nulla, il dataset iniziale viene arricchito di ulteriori punti non appartenenti alla superficie $\mathscr{S}$ sui quali viene imposto ad f di assumere valori non nulli:

$$\begin{aligned} f(\boldsymbol{x}_i) &= 0 & i &= 1, \ldots, n & &\text{(punti } \textit{on surface}\text{)}, \\ f(\boldsymbol{x}_i) &= d_i \neq 0 & i &= n+1, \ldots, N & &\text{(punti } \textit{off surface}\text{)}, \end{aligned}$$

Una funzione semplicemente calcolabile che soddisfi questi vincoli è la funzione *distanza segnata*, che definisce i valori d_i come la distanza fra il corrispondente punto off-surface $\boldsymbol{x}_i$ ed il più vicino punto della superficie. I punti off-surface vengono generati come proiezione dei punti on surface lungo

le rispettive normali: per motivi di robustezza, da ogni punto on surface $\boldsymbol{x}_i$ si ricavano due punti off surface, entrambi lungo la direzione della normale $\vec{\boldsymbol{n}}_i$ ad $\boldsymbol{x}_i$ ma vincolati a giacere uno all'interno e l'altro all'esterno della superficie. Per evitare che i punti off surface così generati intersechino altre porzioni della superficie, è sufficiente garantire che il punto della superficie più vicino al nuovo punto off surface sia quello attraverso cui tale punto è stato generato (figura 2.8).

Una volta arricchito il dataset con i nuovi punti off surface e calcolato per ogni punto la distanza segnata dalla superficie, l'algoritmo cerca di individuare una funzione interpolante $s(\boldsymbol{x})$ attraverso la quale approssimare la funzione distanza segnata $f(\boldsymbol{x})$, vale a dire:

$$s(\boldsymbol{x}_i) = f(\boldsymbol{x}_i) = f_i \qquad i = 1, \cdots, N. \tag{2.2}$$

Tale interpolante viene scelto nello spazio $BL^{(2)}(\mathbb{R}^3)$ di Beppo-Levi delle distribuzioni su $\mathbb{R}^3$ con derivate seconde quadrate integrabili; poiché in questo spazio il problema appena delineato ha più soluzioni, è necessario definire lo spazio affine degli interpolanti:

$$S = \{s \in BL^{(2)}(\mathbb{R}^3) : s(\boldsymbol{x}_i) = f_i, \quad i = 1, \cdots, N\}.$$

All'interno di questo spazio è definita la seminorma:

$$\begin{aligned} \|s\|^2 = \int_{\mathbb{R}^3} \Bigg(\left(\frac{\partial^2 s(\boldsymbol{x})}{\partial x^2}\right)^2 + \left(\frac{\partial^2 s(\boldsymbol{x})}{\partial y^2}\right)^2 + \left(\frac{\partial^2 s(\boldsymbol{x})}{\partial z^2}\right)^2 + \\ 2\left(\frac{\partial^2 s(\boldsymbol{x})}{\partial x \partial y}\right)^2 + 2\left(\frac{\partial^2 s(\boldsymbol{x})}{\partial x \partial z}\right)^2 + 2\left(\frac{\partial^2 s(\boldsymbol{x})}{\partial y \partial z}\right)^2 \Bigg) dx \end{aligned} \tag{2.3}$$

che è una misura dell'energia, o della "regolarità", delle funzioni: funzioni con seminorma minore sono più smooth, o più regolari, di funzioni con seminorma maggiore. È stato dimostrato che all'interno di questo spazio l'interpolante di seminorma minima è della forma:

$$s^\star(\boldsymbol{x}) = \arg\min_{s \in S} \|s\| = p(\boldsymbol{x}) + \sum_{i=1}^{N} \lambda_i \|\boldsymbol{x} - \boldsymbol{x}_i\|, \tag{2.4}$$

in cui p è un polinomio di grado basso, i coefficienti λ_i sono valori reali e $\|\cdot\|$ è la standard norma euclidea in $\mathbb{R}^3$. Questa equazione altro non è che una particolare esempio di *radial basis function*, funzioni dalla generica forma:

$$s(\boldsymbol{x}) = p(\boldsymbol{x}) + \sum_{i=1}^{N} \lambda_i \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|), \tag{2.5}$$

in cui p è un polinomio di grado limitato e ϕ è una funzione definita in $\mathbb{R}^+$ detta *basis function*, generalmente non limitata e di supporto non compatto; in quest'ambito, i coefficienti λ_i ed i punti $\boldsymbol{x}_i$ sono detti rispettivamente i *pesi* ed i *centri* della RBF. La scelta dei pesi λ_i in equazione 2.4 non può essere fatta arbitrariamente ma è vincolata alla richiesta $s^\star \in BL^{(2)}(\mathbb{R}^2)$, la quale implica la cosiddettà condizione di ortogonalità:

$$\sum_{i=1}^{N} \lambda_i = \sum_{i=1}^{N} \lambda_i x_i = \sum_{i=1}^{N} \lambda_i y_i = \sum_{i=1}^{N} \lambda_i z_i = 0.$$

Queste condizioni imposte sui coefficienti implicano che, se m è il grado del polinomio $p(\boldsymbol{x}_i)$ in equazione 2.5, allora deve valere

$$\sum_{i=1}^{N} \lambda_i q(\boldsymbol{x}_i) = 0 \qquad \forall \text{ polinomio } q(\boldsymbol{x}_i) : \deg(q) \leqslant m \tag{2.6}$$

Le condizioni sull'interpolazione espresse in 2.2 insieme alla precedente condizione di ortogonalità permettono di costruire un sistema lineare da risolvere rispetto ai coefficienti che caratterizzano la RBF.

Se $\{p_1, \ldots, p_\ell\}$ indicano una base per i polinomi di grado al più m e $c = (c_1, \ldots, c_\ell)$ è una base che permette di esprimere il polinomio p come combinazione lineare di questa base, allora le equazioni 2.2 e 2.6 possono essere riscritte nella seguente forma matriciale:

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = B \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix},$$

in cui

$$A_{i,j}=\phi(\|\boldsymbol{x}_i-\boldsymbol{x}_j\|), \qquad i,j=1,\ldots,N$$
$$P_{i,j}=p_j(\boldsymbol{x}_i), \qquad i=1,\ldots,N, \quad j=1,\ldots,\ell$$

La risoluzione di questo sistema lineare determina i pesi λ_i ed i centri $\boldsymbol{c}_i$ della RBF e, di conseguenza, l'interpolante $s^\star$. È doveroso sottolineare che la matrice così costruita risulta essere tanto più mal condizionata quanto più il numero N dei punti è grande e che quindi il risultato è sicuramente affetto da errore.

In presenza di rumore nei dati i vincoli espressi con l'equazione 2.2 sono troppo stringenti e sarebbe preferibile trovare una funzione che rispetto a quella appena definita avesse seminorma $\|s\|^2$ inferiore e, di conseguenza, avesse un comportamento più smooth; il problema viene così riformulato:

$$s^\star(\boldsymbol{x}_i) = \min_{s\in BL^{(2)}(\mathbb{R}^2)} \rho\|s\|^2 + \frac{1}{N}\sum_{i=1}^{N}\big(s(\boldsymbol{x}_i)-f_i\big)^2,$$

dove $\|\cdot\|$ è la norma come definita nell'equazione 2.3 ed il parametro $\rho \geqslant 0$ bilancia fra fedeltà ai dati ed il grado di regolarità della funzione. L'interpolante $s^\star$ per questo nuovo problema è della stessa forma mostrata in precedenza, mentre è stato dimostrato che il vettore $(\lambda^T, c^T)^T$ è ora soluzione del problema:

$$\begin{pmatrix} A-8N\pi\rho I & P \\ P^T & 0 \end{pmatrix}\begin{pmatrix} \lambda \\ c \end{pmatrix} = B\begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix},$$

in cui le matrici A e P sono come definite in precedenza.

Una volta eseguito il *fitting* dell'RBF, essa viene valuta per ricavare l'isosuperficie dell'oggetto. Per riuscire a velocizzare quest'ultima fase la valutazione dell'RBF è effettuata attraverso il cosiddetto *Fast Multipole Method* (o FMM), un metodo approssimativo impiegabile in tutte quelle circostanze in cui non è necessaria una precisione infinitesima. I centri $\boldsymbol{c}_i$ della RBF sono clusterizzati in maniera gerarchica e l'approssimazione della RBF in un dato

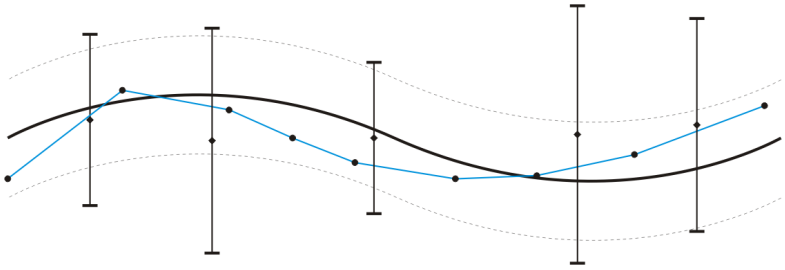


Figura 2.9: *Rappresentazione dei parametri coinvolti nei metodi fast fitting. I punti rappresentano i centri di valutazione generati in output, mentre i rombi indicano i nodi di valutazione. Le due linee tratteggiate indicano l'evaluation accuracy, la linea nera più spessa l'RBF ed infine la linea blu la superfice generata in output.*

punto è calcolata attraverso espansioni *far field* e *near field* dei centri all'interno di un dato cluster. A seconda della distanza fra il punto di valutazione ed i centri della RBF vengono impiegate valutazioni approssimate o dirette: l'utilizzo di questa tecnica non preclude in nessun modo l'accuratezza del modello ricostruito, mentre consente di abbassare notevolmente il tempo di calcolo rispetto all'utilizzo di sole valutazioni dirette. La precisione di questo metodo è regolato attraverso due parametri, il *fitting accuracy* e l'*evaluation accuracy*: il primo rappresenta la massima deviazione fra il valore della RBF ed il valore specificato ai nodi d'interpolazione, mentre il secondo specifica la precisione con cui l'RBF viene valutata.

Un'altra tecnica sempre mirata alla riduzione dei tempi di valutazione della RBF consiste nella riduzione dei suoi centri: le RBF utilizzano in genere tutti i punti nel dataset sia come nodi di interpolazione sia come centri della RBF. Tuttavia è possibile approssimare la superficie con un prefissato grado di precisione pur utilizzando un numero di centri $\boldsymbol{c}_i$ decisamente minore alla dimensione del dataset. Tale obiettivo può essere conseguito attraverso un algoritmo *greedy* che, partendo da un sottoinsieme di punti scelti

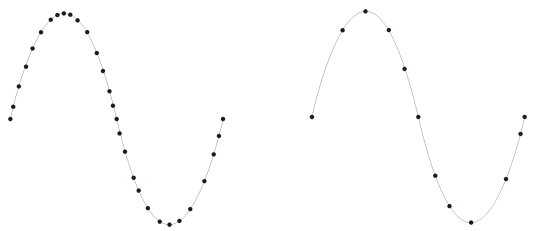


Figura 2.10: *L'applicazione della tecnica per la riduzione dei centri di valutazione dell'RBF.*

in modo random fra quelli all'interno del dataset, esegue iterativamente il fitting dell'RBF fino al raggiungimento della prefissata *fitting accuracy*: tale parametro può essere o un parametro globale, che stima l'errore complessivo accumulato durante la ricostruzione della superficie, oppure un parametro caratteristico di ogni punto che vincola localmente il modello ad una determinata precisione.

2.2.3 Implicit Moving Least Squares Surface

La descrizione implicita di una superficie trova impiego oltre che durante la fase di fusione anche durante quella di postprocessing, dove le sue qualità possono essere sfruttate a garanzia di determinate proprietà nel modello ricostruito. Sebbene questo capitolo sia destinato alla trattazione degli algoritmi di fusione, verrà ora descritto un metodo che sfrutta la validità delle rappresentazioni implicite per ripulire un modello tridimensionale dalle più comuni imperfezioni; la normale applicazione di questo algoritmo quindi sarebbe a rigore all'interno della fase di postprocessing: abbiamo ritenuto però che la sua trattazione in questa sede, giustificata tra l'altro anche dalla possibilità di considerare un insieme di range map come un modello tridimensionale da aggiustare, aiuti a comprendere la validità e la versatilità delle rappresentazioni implicite.

L'obiettivo di questo algoritmo [32] è la definizione di una superficie implicita che interpoli (o approssimi) un modello poligonale di input: il grado con cui la superficie implicita si adatta al modello iniziale è regolato da un parametro detto *minimum feature size*, che specifica la dimensione delle più piccole caratteristiche da preservare; la superficie generata dall'algoritmo, sia essa approssimante o interpolante, elimina le auto-intersezioni e si estende sui buchi del modello originale, generandone una descrizione *watertight*.

Alla base della procedura di generazione della superficie implicita è il metodo di interpolazione per dati sparsi noto come *moving least squares* o MLS. A differenza del metodo di approssimazione dei minimi quadrati, l'MLS consente di generare una funzione il cui comportamento è influenzato anche dal punto di valutazione $\boldsymbol{p}$. Il sistema da risolvere è:

$$\begin{bmatrix} w(\boldsymbol{p},\boldsymbol{x}_1) & & \\ & \ddots & \\ & & w(\boldsymbol{p},\boldsymbol{x}_n) \end{bmatrix} \begin{bmatrix} \mathbf{b}^T(\boldsymbol{x}_1) \\ \vdots \\ \mathbf{b}^T(\boldsymbol{x}_n) \end{bmatrix} \mathbf{c} = \begin{bmatrix} w(\boldsymbol{p},\boldsymbol{x}_1) & & \\ & \ddots & \\ & & w(\boldsymbol{p},\boldsymbol{x}_n) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_n \end{bmatrix}, \quad (2.7)$$

dove $\mathbf{b}(\boldsymbol{x}_i)$ è il vettore delle basi utilizzate per la definizione della nuova funzione, $\mathbf{c}$ è il vettore incognita dei coefficienti, ϕ_i è il valore da approssimare ai punti $\boldsymbol{x}_i$ e $w(\boldsymbol{p},\boldsymbol{x}_i) = w(\|\boldsymbol{p}-\boldsymbol{x}_i\|)$ è una qualche *weighting function* basata sulla distanza. Il comportamento della superficie implicita dipende strettamente dalla weight function w: in generale una funzione che tende all'infinito per distanze prossime allo zero genera una superficie implicita interpolante. Una funzione che seppur semplice da calcolare produce risultati buoni almeno quanto scelte computazionalmente più costose è:

$$w(\|\boldsymbol{p}-\boldsymbol{x}_i\|) = w(d_i) = \frac{1}{{d_i}^2+\epsilon^2} \quad (2.8)$$

in cui il parametro ϵ, sul cui significato si tornerà successivamente, garantisce un certo livello di controllo sul comportamento della superficie. Scrivendo l'equazione 2.7 in forma matriciale ed esplicitando la dipendenza dal punto

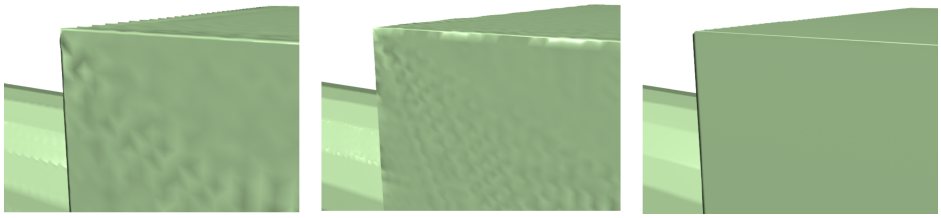


Figura 2.11: *Se i vincoli sui poligoni fossero espressi semplicemente attraverso un insieme di punti sparsi su di essi, la superficie presenterebbe delle increspature (a sinistra e al centro, con diversi valori di densità); a destra invece la superficie generata estendendo continuamente sui poligoni i vincoli ai vertici.*

$\boldsymbol{p}$ di valutazione, si ottiene:

$$\mathbf{W}(\boldsymbol{p})\,\mathbf{B}\,\mathbf{c}(\boldsymbol{p}) = \mathbf{W}(\boldsymbol{p})\,\boldsymbol{\Phi}; \tag{2.9}$$

il suo sistema di equazioni normali è:

$$\mathbf{B}^T\,(\mathbf{W}(\boldsymbol{p}))^2\,\mathbf{B}\,\mathbf{c}(\boldsymbol{p}) = \mathbf{B}^T\,(\mathbf{W}(\boldsymbol{p}))^2\,\boldsymbol{\Phi} \tag{2.10}$$

da cui è possibile infine ricavare la funzione approssimante:

$$f(\boldsymbol{p}) = \mathbf{b}^T(\boldsymbol{p})\big(\mathbf{B}^T\,(\mathbf{W}(\boldsymbol{x}))^2\,\mathbf{B}\big)^{-1}\,\mathbf{B}^T(\mathbf{W}(\boldsymbol{x}))^2\,\boldsymbol{\Phi} \tag{2.11}$$

I vincoli espressi nell'equazione 2.7 si limitano a descrivere il comportamento della superficie interpolante sull'insieme dei vertici del modello; in realtà è possibile forzare la superficie implicita a stare il più "aderente" possibile ai poligoni del modello semplicemente estendendo l'insieme dei vincoli. In linea di principio vincolare la superficie ad un poligono significa vincolare la superficie ad un insieme infinito di punti sparsi sul poligono stesso: l'espressione 2.10 delle equazioni normali può essere esplicitata come una sommatoria su un insieme di vincoli ai punti:

$$\Big(\sum_{i=1}^{n} w^2(\boldsymbol{p},\boldsymbol{x}_i)\,\mathbf{b}(\boldsymbol{x}_i)\,\mathbf{b}^T(\boldsymbol{x}_i)\Big)\,\mathbf{c}(\boldsymbol{p}) = \sum_{i=1}^{n} w^2(\boldsymbol{p},\boldsymbol{x}_i)\,\mathbf{b}(\boldsymbol{x}_i)\,\phi_i, \tag{2.12}$$

e poiché gli elementi delle due sommatorie altro non sono che vincoli sulle facce dei poligoni, essi possono essere estesi *continuamente* attraverso integrali definiti sui poligoni stessi; se $\mathscr{T} = \{\tau_1, \ldots, \tau_t\}$ è l'insieme dei poligoni nel modello, le due sommatorie esplicite in 2.12 possono essere sostituite da integrali sui poligoni:

$$\Big(\sum_{k=1}^{t} w^2(\boldsymbol{p}, \boldsymbol{x})\, \widetilde{A}_k \Big)\, \mathbf{c}(\boldsymbol{p}) = \sum_{k=1}^{t} w^2(\boldsymbol{p}, \boldsymbol{x})\, \widetilde{a}_k, \tag{2.13}$$

dove $\widetilde{A}_k$ e $\widetilde{a}_k$ sono così definite:

$$\widetilde{A}_k = \int_{\tau_k} \mathbf{b}(\boldsymbol{p})\, \mathbf{b}^T(\boldsymbol{p})\, \mathrm{d}\boldsymbol{p} \tag{2.14}$$

$$\widetilde{a}_k = \int_{\tau_k} \mathbf{b}(\boldsymbol{p})\, \phi_k\, \mathrm{d}\boldsymbol{p}, \tag{2.15}$$

in cui la variabile d'integrazione è il punto $\boldsymbol{p}$ che varia sul poligono mentre ϕ_k è un parametro che può rimanere costante o variare polinomialmente sul poligono stesso.

Sebbene la superficie implicita, per come è stata finora descritta, sia vincolata ad assumere determinati valori sia ai vertici che ai poligoni del modello tridimensionale, tuttavia il suo comportamento in prossimità della superficie non è soddisfacente, in quanto non vincolato lungo una particolare direzione. Una delle grandi novità introdotte da questo algoritmo è stata la possibilità di forzare la funzione interpolante non semplicemente ad un determinato valore ma a comportarsi, nell'interno di un poligono, come una prestabilita funzione: il metodo dell'MLS viene quindi adoperato non per interpolare tra valori costanti ma bensì fra funzioni associate ai poligoni. Se $\vec{\boldsymbol{n}}_k$ è la normale associata al poligono τ_k, la funzione $\xi_k(\boldsymbol{p})$, che descrive il comportamento dell'interpolante in prossimità del poligono τ_k, viene definita come:

$$\begin{aligned} \xi_k(\boldsymbol{p}) &= \phi_k + (\boldsymbol{p} - \boldsymbol{q}_k)^T \cdot \vec{\boldsymbol{n}}_k \\ &= \psi_{0k} + \psi_{xk} x + \psi_{yk} y + \psi_{zk} z, \end{aligned}$$

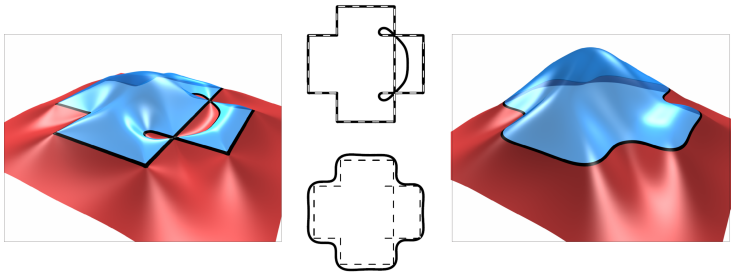


Figura 2.12: *Un esempio bidimensionale che illustra le differenze fra la superficie interpolante (a sinistra) e quella approssimante (a destra). Al centro, le linee tratteggiate rappresentano i vincoli, mentre la linea più spessa è il contorno della figura. Ai lati una rappresentazione height-field della relative funzioni.*

dove $\boldsymbol{q}_k$ è un arbitrario punto sul poligono τ_k e $\psi_{0k}, \psi_{xk}, \psi_{yk}$ e ψ_{zk} sono i coefficienti del polinomio risultante. L'interpolazione dei valori di queste funzioni si riduce dunque ad interpolare i coefficienti ψ, esattamente come nell'equazione 2.7 era necessario interpolare i valori costanti ϕ_k.

La funzione generata a partire dall'insieme dei vincoli ai vertici, alle facce ed alle normali implicitamente descrive la superficie del modello tridimensionale: la fedeltà di tale descrizione è regolata attraverso il parametro ϵ all'interno della weighting function 2.8. Quando ϵ è impostata a zero, la funzione interpolerà esattamente i valori esplicitati ai vincoli e dunque la superficie da essa generabile includerà tutti i vertici ed i poligoni del modello di partenza ma sarà priva di eventuali imperfezioni, quali buchi, che verranno sostituiti da nuovi poligoni, ed auto-intersezioni, al cui posto saranno presenti delle sorte di selle (figura 2.12). Se invece ϵ è impostato ad un valore diverso da zero, la weighting function non ha più singolarità a zero e la superficie IMLS approssimerà i vincoli specificati: in questo caso il parametro ϵ può essere interpretato come la *minimum feature size*, nel senso che le caratteristiche dell'oggetto di dimensioni minori di ϵ verranno escluse dalla superficie

interpolante.

È evidente che l'applicazione dell'algoritmo a problemi reali non può prescindere dall'utilizzo della *fast evaluation*, in quanto altrimenti la complessità di ogni valutazione sarebbe lineare nel numero di vincoli, una complessità assolutamente insostenibile anche per input di dimensioni non eccessive. La causa di tale complessità è prevalentemente imputabile al calcolo della equazione 2.13. In effetti ciò che cambia all'interno di questa equazione è solamente il valore della weighting function; per di più, avendo in genere queste funzioni un supporto compatto, il cambiamento è sensibile solamente in prossimità della superficie e decresce all'aumentare della distanza dalla superficie. Ricorrendo ad una struttura gerarchica all'interno della quale memorizzare per ogni poligono i valore degli integrali 2.14, è possibile una valutazione approssimata di quelle porzioni della superficie lontana dal punto di valutazione, velocizzando quindi nel complesso l'intero processo di estrazione dell'isosuperficie.

2.2.4 Multilevel Partition of Unity Implicit surface

Gli ultimi due metodi descritti, quelli cioè basati su RBF e su MLS, seppur diversi per formulazione, sono riconducibili ad una stessa famiglia di algoritmi che descrivono implicitamente la superficie dell'oggetto attraverso un'unica funzione globale. Utilizzando delle funzioni di approssimazione locale alla superficie e delle weighting function per effettuare il blending di quest'ultime, l'algoritmo che verrà ora descritto [25] riesce a costruire una rappresentazione implicita locale del dataset di input. L'idea alla base di questo algoritmo è di costruire localmente l'approssimazione della funzione distanza segnata dalla superficie attraverso un procedura adattiva controllata dall'errore di approssimazione: la superficie viene quindi approssimata dall'insieme zero $Z(f)$ della funzione distanza segnata ricostruita.

Il blending fra le funzioni locali è effettuato attraverso delle weighting function generate secondo l'approccio *partition of unity*, un metodo in genere utilizzato per integrare approssimazioni definite localmente in un'unica approssimazione globale e che non casualmente è riconducibile [7] all'approccio MLS adottato dall'algoritmo precedente. In linea di massima, l'approccio partition of unity prevede di partizionare in diversi sottoinsiemi il dominio dei dati, approssimare individualmente i dati in ogni sottodominio e quindi combinare insieme le soluzioni locali utilizzando dei pesi la cui somma su tutto il dominio deve valere uno. Più precisamente, siano Ω un dominio chiuso in uno spazio euclideo e $\{\varphi_i\}$ un insieme di funzioni non-negative con supporto compatto tali che

$$\sum_i \varphi_i \equiv 1 \text{ su } \Omega$$

Se ad ogni sottodominio $\text{supp}(\varphi_i)$ viene associato un insieme F_i di funzioni di approssimazione locale, allora una generica funzione $f(\boldsymbol{p})$ definita su Ω può essere approssimata, attraverso sottoinsiemi $Q_i \in F_i$, come:

$$f(\boldsymbol{p}) \approx \sum_i \varphi_i(\boldsymbol{p})\, Q_i(\boldsymbol{p}).$$

Le funzioni partition of unity (o MPU) $\{\varphi_i\}$ possono cioè essere generate a partire da un insieme di funzioni non negative e con supporto compatto $\{w_i(\boldsymbol{p})\}$ che verifichino la condizione $\Omega \in \bigcup_i \text{supp}(w_i)$ semplicemente imponendo:

$$\varphi_i(\boldsymbol{p}) = \frac{w_i(\boldsymbol{p})}{\sum_{j=1}^{n} w_j(\boldsymbol{p})}$$

Diverse scelte per le funzioni w_i consentono di generare superfici implicite che esibiscono comportamenti diversi: nel caso sia richiesto che la superficie approssimi i punti nel dataset, allora la weighting function w_i viene costruita attraverso la B-spline quadratica $b(t)$:

$$w_i(\boldsymbol{p}) = b\left(\frac{3\|\boldsymbol{p} - \boldsymbol{c}_i\|}{2\rho_i}\right), \tag{2.16}$$

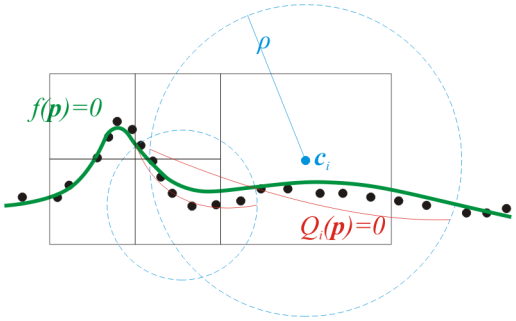


Figura 2.13: *La suddivisione del volume è adattiva e l'insieme dei punti all'interno di ogni cella è descritto con una funzione $Q_i(\boldsymbol{p})$ di approssimazione locale; il blending per mezzo della weighting function delle varie $Q_i(\boldsymbol{p})$ individua infine la funzione $f(\boldsymbol{p})$ che descrive la superficie.*

di centro $\boldsymbol{c}_i$ e di supporto sferico di raggio ρ; se invece è richiesta una superficie interpolante, allora w_i è definita come:

$$w_i(\boldsymbol{p}) = \left[\frac{(\rho_i - \|\boldsymbol{p} - \boldsymbol{c}_i\|)_+}{\rho_i \|\boldsymbol{p} - \boldsymbol{c}_i\|}\right]^2, \quad \text{dove } (a)_+ = \begin{cases} a & \text{se } a > 0 \\ 0 & \text{altrimenti} \end{cases}$$

Le weighting function così definite sono quindi utilizzare per effettuare il blending fra le funzioni che localmente approssimano la superficie: queste ultime sono associate alle celle di un octree e sono definite come approssimazione dei punti all'interno della cella. La suddivisione del volume è adattiva nel senso che la profondità dell'octree varia localmente in funzione della densità di campionamento in quel punto (figura 2.13). Si consideri una generica cella dell'octree generata durante il processo di suddivisione dello spazio e siano $\boldsymbol{c}$ il suo centro e d la lunghezza della sua diagonale. Il raggio ρ del supporto per la weighting function 2.16 è definito proporzionalmente alla diagonale della cella, cioè $\rho = \alpha d$: valori maggiori di α producono superfici interpolanti e approssimanti più fedeli al prezzo di un maggior tempo di calcolo. Per ognuna delle celle così generate, viene definita una funzione $Q(\boldsymbol{p})$, la *local shape function*, con la quale attraverso il metodo dei minimi quadrati

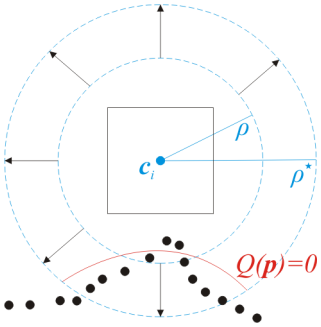


Figura 2.14: *Se il numero di punti all'interno del supporto di una weighting function non è sufficiente per definire una robusta approssimazione locale, il raggio del supporto viene espanso fino ad includere almeno n_{min} punti.*

vengono approssimati i punti all'interno della cella. Il processo di suddivisione delle celle e di generazione di un'approssimazione locale si arresta solamente quando la stima dell'errore d'approssimazione

$$\epsilon = \max_{\|\boldsymbol{p}_i - \boldsymbol{c}_i\|} \frac{|Q(\boldsymbol{p}_i)|}{|\nabla Q(\boldsymbol{p}_i)|} \tag{2.17}$$

diventa minore della soglia ϵ_0 specificata dall'utente.

Talvolta, specialmente quando la densità di campionamento non è uniforme, il sfera di raggio ρ associata ad una cella non contiene un numero di punti sufficienti per una robusta definizione della local shape function $Q(\boldsymbol{p})$: in questo caso, il raggio della sfera viene iterativamente espanso fino al raggiungimento di un prefissato numero minimo $n_{\min}$ di punti (figura 2.14); i punti inclusi all'interno di questo nuovo raggio sono quindi utilizzati per generare la funzione di approssimazione locale.

La strategia di approssimazione locale dipende sia dal numero di punti contenuti nella sfera associata ad una cella sia alla distribuzione delle normali a questi punti. Siano $\mathscr{P}$ l'insieme dei punti contenuti all'interno della sfera di raggio ρ centrata nel centro $\boldsymbol{c}$ di una cella, $\mathscr{N}$ l'insieme delle normali

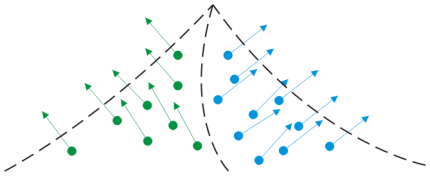


Figura 2.15: *Clusterizzazione delle normali: l'orientazione delle normali ai punti consente di dedurre la presenza di due piani la cui intersezione forma uno spigolo; l'intersezione di tre piani avrebbe invece individuato un angolo.*

dei punti in $\mathscr{P}$, $\vec{\boldsymbol{n}}$ la normale assegnata al punto $\boldsymbol{c}$ calcolata come media aritmetica pesata normalizzata delle normali in $\mathscr{N}$ e ϑ l'angolo di deviazione massima fra $\vec{\boldsymbol{n}}$ e le normali in $\mathscr{N}$.

- Se il numero di punti in $\mathscr{P}$ è maggiore di $2n_{\min}$ ed inoltre $\vartheta \geqslant \frac{\pi}{2}$, la superficie viene approssimata attraverso una generica quadrica tridimensionale, che fornisce una buona approssimazione per porzioni estese di superficie potenzialmente anche non limitate.

- Se invece il numero di punti in $\mathscr{P}$ è sì maggiore di $2n_{\min}$ ma $\vartheta \leqslant \frac{\pi}{2}$, la superficie in questa cella viene approssimata attraverso un polinomio quadratico in due variabili espresso in coordinate locali.

- Se infine $\mathscr{P}$ contiene meno di $2n_{\min}$ punti, è necessario effettuare altri test, prevalentemente basati sulla clusterizzazione delle normali, per verificare se la superficie contenga uno spigolo o un angolo (vedi figura 2.15).

È giusto far osservare che è inutile verificare la presenza di spigoli o angoli nel caso in cui la sfera associata alla cella contenga più di $2n_{\min}$ punti. Le *sharp features* infatti influenzano il processo di suddivisione del volume attraverso la misura 2.17: se i punti all'interno di una cella avessero descritto una

sharp feature, allora la cella sarebbe stata divisa e tale feature sarebbe stata approssimata attraverso le diverse local shape function associate alle celle generate a partire da questa suddivisione.

2.2.5 Point Set surface

Una tecnica di recente introduzione in computer graphics e che sta iniziando a raccogliere consensi presso diversi centri di ricerca è quella di modellare la superficie degli oggetti tramite un insieme molto denso di punti, detto *point-set*, anziché attraverso mesh simpliciali. Seguendo questo approccio sono stati sviluppati diversi metodi, accomunati sotto il nome di *costruzioni meshless*, orientati alla definizione della superficie implicata da una nuvola di punti: ad accomunare questi metodi è la presenza di una procedura attraverso la quale proiettare punti arbitrariamente scelti nello spazio sulla superficie individuata dalla nuvola di punti del dataset.

La prima formulazione di superficie point-set fu la MLS surface [20], ampiamente ripresa da diversi metodi basati sulla stessa rappresentazione. La superficie MLS per una nuvola di punti $\mathscr{C} \in \mathbb{R}^3$ è definita come l'insieme dei punti stazionari di una qualche funzione $f : \mathbb{R}^3 \to \mathbb{R}^3$, dove f è la procedura utilizzata per proiettare i punti in un intorno di $\mathscr{C}$ sulla superficie da questi individuata: secondo questa definizione quindi un punto $\boldsymbol{p} \in \mathbb{R}^3$ appartiene alla MLS surface solamente quando $f(\boldsymbol{p}) = \boldsymbol{p}$. Più precisamente, dati la nuvola di punti $\mathscr{C}$ ed un punto $\boldsymbol{p}$ in prossimita di $\mathscr{C}$, l'energia associata al piano H di normale $\vec{\boldsymbol{n}}$ passante per il punto $\boldsymbol{q} = \boldsymbol{p} + t\vec{\boldsymbol{n}}$, dove t è la distanza di $\boldsymbol{p}$ dal piano, è definita come:

$$e_{\mathrm{MLS}}(\vec{\boldsymbol{n}}, t) = \sum_{\boldsymbol{x}_i \in \mathscr{C}} \big(\vec{\boldsymbol{n}} \cdot \boldsymbol{x}_i - (\boldsymbol{p} + t\vec{\boldsymbol{n}}) \cdot \vec{\boldsymbol{n}}\big)^2 \vartheta\big(\boldsymbol{p} + t\vec{\boldsymbol{n}}, \boldsymbol{x}_i\big).$$

Questa energia misura la qualità dell'approssimazione del piano alla nuvola di punti, pesando il contributo di ogni punto $\boldsymbol{x}_i$ con la sua distanza

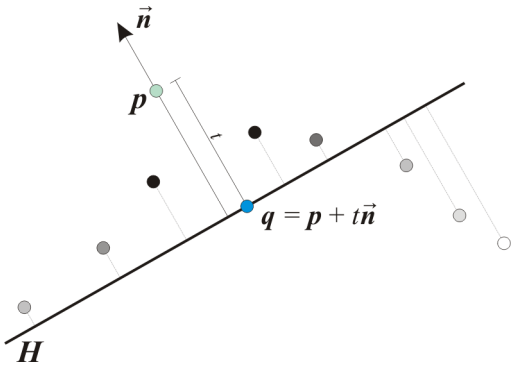


Figura 2.16: *L'energy function e_{MLS} è calcolata in termini delle distanze pesate dal punto di input $\boldsymbol{p}$ al piano passante per $\boldsymbol{q} = \boldsymbol{p} + t\vec{\boldsymbol{n}}$ ed orientato in direzione $\vec{\boldsymbol{n}}$. I pesi dei punti, indicati dalle loro tonalità di grigio, sono funzione della loro distanza da $\boldsymbol{p}$.*

dal punto $\boldsymbol{q}$ attraverso la weighting function ϑ; quest'ultima funzione è necessario che sia monotona decrescente, in genere si può usare una funzione gaussiana della distanza:

$$\vartheta\big(\boldsymbol{q}, \boldsymbol{x}_i\big) = e^{-\frac{\|\boldsymbol{q}-\boldsymbol{x}_i\|^2}{h^2}},$$

con h un constant scaling factor e $\|\cdot\|$ la norma euclidea. Il minimo locale dell'energy function e_{MLS} in $\mathbb{S}^2 \times \mathbb{R}^3$, dove $\mathbb{S}^2$ indica lo spazio delle direzioni, si ha su un insieme discreto di coppie $(\vec{\boldsymbol{n}}, t)$, ognuna delle quali identifica univocamente un punto $\boldsymbol{q} = \boldsymbol{p} + t\vec{\boldsymbol{n}}$: di tali punti $f(\boldsymbol{p})$ è definita per costruzione come il punto $\boldsymbol{q}$ più vicino a $\boldsymbol{p}$: l'insieme dei punti stazionari del mapping f definisce implicitamente la superficie MLS.

Una semplice riformulazione della definizione appena fornita dell'energy function permette di evidenziarne alcune sue interessanti proprietà: la nuova formulazione renderà e_{MLS} indipendente dal punto $\boldsymbol{p}$ e consentirà di fornire una diversa caratterizzazione delle superficie MLS. Innanzitutto, l'energy function e_{MLS} può essere espressa come funzione di $\vec{\boldsymbol{n}}$ e $\boldsymbol{q} = \boldsymbol{p} + t\vec{\boldsymbol{n}}$ anziché

come funzione di $\vec{\boldsymbol{n}}$ e t; quindi, notando che il piano ed i pesi individuati dai parametri $(\boldsymbol{q}, \vec{\boldsymbol{n}})$ sono gli stessi di quelli individuati dai parametri $(\boldsymbol{q}, -\vec{\boldsymbol{n}})$, possiamo esprimere l'energy function come funzione del punto $\boldsymbol{p}$ e di un vettore direzionale *non orientato* $\bar{\boldsymbol{n}}$; la nuova definizione[1] di e_{MLS} è cioè:

$$e_{\text{MLS}}(\boldsymbol{q}, \bar{\boldsymbol{n}}) = \sum_{\boldsymbol{x}_i \in \mathscr{C}} \big((\boldsymbol{x}_i \cdot \bar{\boldsymbol{n}}) - (\boldsymbol{q} \cdot \bar{\boldsymbol{n}})\big)^2 \vartheta(\boldsymbol{q}, \boldsymbol{x}_i).$$

Il dominio della energy function e_{MLS} cos definita è ora $\mathbb{P}^2 \times \mathbb{R}^3$, dove $\mathbb{P}^2$ è lo spazio delle direzioni non orientate, che è un dominio a cinque dimensioni; tuttavia l'utilizzo della nuova notazione consente minimizzare l'energia solamente nel sottoinsieme tridimensionale $J_{\boldsymbol{p}}$ dei valori $(\boldsymbol{q}, \bar{\boldsymbol{n}})$ tali che $\boldsymbol{q} = \boldsymbol{p} + t\vec{\boldsymbol{n}}$, per qualche valore $t \in \mathbb{R}^3$ e orientazione $\vec{\boldsymbol{n}}$ di $\bar{\boldsymbol{n}}$. Questo implica che in $J_{\boldsymbol{p}}$ ogni punto $\boldsymbol{q} \in \mathbb{R}^3$ diverso da $\boldsymbol{p}$ è associato alla direzione $\bar{\boldsymbol{n}}$ della linea passante per $\boldsymbol{p}$ e $\boldsymbol{q}$, mentre solo il punto $\boldsymbol{p}$ è associato a tutte le direzioni in $\mathbb{P}^2$. Diversi valori di $\boldsymbol{p}$ generano quindi diversi valori di $f(\boldsymbol{p})$ dato che diverse scelte di $\boldsymbol{p}$ implicano domini $J_{\boldsymbol{p}}$ diversi in cui minimizzare l'energia e_{MLS}.

Rimane infine da definire la funzione generatrice dell'*unoriented vector field*: la direzione associata ad un punto $\boldsymbol{p}$ è la stessa della normale al piano passante per $\boldsymbol{p}$ che risulti la miglior approssimazione alla nuvola di punti $\mathscr{C}$, vale a dire:

$$n(\boldsymbol{p}) = \operatorname{argmin}_{\bar{\boldsymbol{n}}} e_{\text{MLS}}(\boldsymbol{p}, \bar{\boldsymbol{n}})$$

Poiché l'individuazione del punto $\boldsymbol{p}$ determina automaticamente anche la weighting function ϑ, se ne ricava che l'energy function e_{MLS} è una funzione quadratica di $\bar{\boldsymbol{n}}$ e che la direzione di minimo è generalmente unica; questa può essere individuata o sfruttando le normali presenti ai punti o, qualora queste non siano disponibili, attraverso l'individuazione dell'autovalore minimo di una matrice 3×3 (vedi 2.1 a pagina 26).

[1] Seppure il prodotto scalare non sia definito per vettori direzionali non orientati, si manterrà comunque questa notazione: del resto la funzione può essere calcolata attraverso il prodotto scalare standard o con $\vec{\boldsymbol{n}}$ o con $-\vec{\boldsymbol{n}}$.

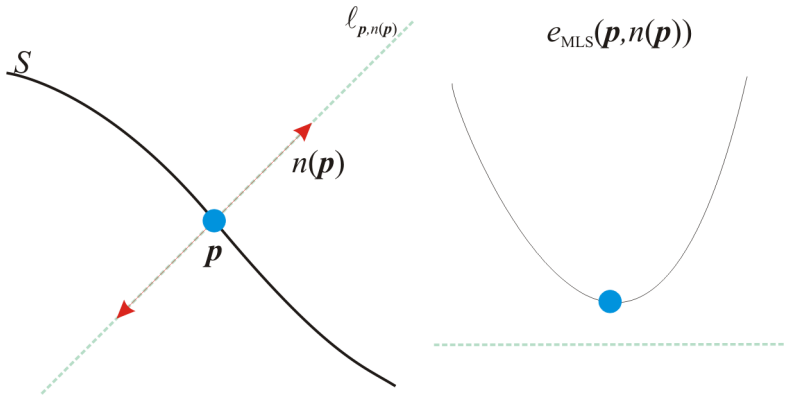


Figura 2.17: *Per verificare se un punto $\boldsymbol{p}$ appartiene alla superficie MLS, si valuta l'energia lungo la linea $\ell_{\boldsymbol{p},n(\boldsymbol{p})}$. Mantenendo costante $n(\boldsymbol{p})$, si valutano le varie posizioni $\boldsymbol{q}$ lungo tale linea: se $\boldsymbol{p}^\star = \boldsymbol{p}$ è un minimo locale di $e_{MLS(\boldsymbol{p}^\star,n(\boldsymbol{p}))}$, allora $\boldsymbol{p}$ appartiene alla superficie MLS.*

Le precedenti definizioni consentono di dare una caratterizzazione esplicita delle superfici MLS, permettendo quindi di determinare se un punto $\boldsymbol{p} \in \mathbb{R}^3$ appartiene o meno alla superficie MLS. Sia $\ell_{\boldsymbol{p},n(\boldsymbol{p})}$ la retta passante per $\boldsymbol{p}$ con direzione $n(\boldsymbol{p})$, e sia $\operatorname{arglocalmin}_{\boldsymbol{q}}$ la funzione che restituisce l'insieme dei punti di minimo locale di una funzione di variabile $\boldsymbol{q}$: allora la superficie MLS consiste di tutti e soli quei punti $\boldsymbol{p}$ per i quali $n(\boldsymbol{p})$ è ben definita e per cui

$$\boldsymbol{p} \in \operatorname*{arglocalmin}_{\boldsymbol{q} \in \ell_{\boldsymbol{p},n(\boldsymbol{p})}} e_{\mathrm{MLS}}\big(\boldsymbol{q}, n(\boldsymbol{p})\big)$$

Scelte diverse delle due funzioni vector field n ed energy function e generano superfici che esibiscono proprietà diverse: infatti qualsiasi funzione che associa una direzione $\bar{\boldsymbol{n}}$ ad un punto $\boldsymbol{p} \in \mathbb{R}^3$, e qualsiasi funzione $e(\boldsymbol{p}, \bar{\boldsymbol{n}})$ che specifichi una energia per gli elementi in $\mathbb{R}^3 \times \mathbb{P}^2$ possono essere utilizzate per definire una superficie diversa; anzi, qualsiasi funzione $n : \mathbb{R}^3 \to \mathbb{P}^2$ ed

$e : \mathbb{R}^3 \times \mathbb{P}^2 \to \mathbb{R}$ definisce la superficie

$$S = \left\{ \boldsymbol{p} \,|\, \boldsymbol{p} \in \operatorname*{arglocalmin}_{\boldsymbol{q} \in \ell_{\boldsymbol{p}, n(\boldsymbol{p})}} e\big(\boldsymbol{q}, n(\boldsymbol{p})\big) \right\}$$

che viene detta *extremal surface* di e ed n; di conseguenza, la MLS surface altro non è che un caso particolare di extremal surface generata sfruttando la energy function e_{MLS} ed il vector field n_{MLS}.

2.3 Analisi comparativa dei metodi di fusione

Concludiamo questo capitolo con un'analisi comparativa dei metodi di integrazione fino ad ora discussi, per ognuno dei quali cercheremo di evidenziarne sia le qualità sia i punti di maggior debolezza; tale analisi ci tornerà particolarmente utile nei capitoli 3 e 4, quando dovremo giustificare la scelta del metodo implicito di fusione adottato come kernel del nostro algoritmo.

Innanzitutto si può notare che un problema che accomuna tutti gli algoritmi di integrazione non impliciti, vale a dire mesh zippering, crust e ball pivoting, deriva dalla proprietà di utilizzare l'insieme dei punti del dataset direttamente come vertici nella superficie ricostruita: come abbiamo avuto modi di accennare in precedenza, l'input degli algoritmi di fusione non deve mai essere considerato privo di errori, a causa sia del rumore associato ad ogni punto ed imputabile alla fase di campionamento sia degli errori di approssimazione introdotti durante la fase di allineamento. Tali errori si manifestano in genere come rumore distribuito uniformemente su tutto il dataset, mentre talvolta possono anche generare, soprattutto ai bordi delle range map o in presenza di sharp feature, il fenomeno degli *outlier*, vale a dire campioni che giacciono molto lontano dalla superficie. Sebbene in genere il rumore può essere ripulito attraverso una fase di preprocessing del dataset, tuttavia l'utilizzo di algoritmi capaci di gestirlo autonomamente è una garanzia sulla

miglior qualità della superficie ricostruita. Al di là di questo problema comune, ognuno degli algoritmi non-impliciti esibisce poi particolari suoi punti deboli. Il mesh zippering, per il fatto di generare la superficie attraverso un processo di clipping delle varie range map, genera spesso delle triangolarizzazioni di qualità decisamente bassa, come emerge anche dal semplice esempio illustrato in figura 2.1 a pagina 12, e risulta inoltre particolarmente sensibile agli errori di allineamento. Le superfici generate invece col metodo crust risultano almeno da questo punto di vista decisamente più pulite anche per via della sua capacità di generare triangolarizzazioni di densità variabile in funzione della curvatura locale della superficie: questa proprietà viene garantita attraverso l'utilizzo del medial axis e della triangolarizzazione di Delaunay che, se da un lato consentono di generare modelli tridimensionali accurati, dall'altro però relegano l'utilizzo di questo metodo a dataset di dimensioni ridotte. A quest'ultima limitazione non risulta invece vincolato l'algoritmo ball pivoting, che grazie anche all'estensione out-of-core proposta dagli autori [8] consente di ricostruire in tempi del tutto ragionevoli la superficie descritta da dataset composti da milioni di campioni; tuttavia la superficie generata, poiché composta da triangoli i cui vertici sono vincolati a non distare fra loro più del raggio ρ della sfera usata per la sua definizione, può risultare composta da componenti o completamente sconnesse oppure connesse con una triangolarizzazione tutt'altro che smooth.

Gli algoritmi di rappresentazione implicita che abbiamo in precedenza esaminato consentono tutti di generare una superficie che risulta meno sensibile agli errori introdotti nelle precedenti stadi della pipeline di acquisizione: questa proprietà deriva principalmente dall'utilizzo, per la definizione della superficie, della funzione f che seppur generata a partire dai punti del dataset, non risulta vincolata ad interpolarli. Ovviamente, le proprietà esibite dalle superfici ricostruite dai vari algoritmi sono ovviamente una conseguenza delle garanzie offerte della funzioni adottate per la loro generazione. Per

esempio, l'algoritmo IMLS genera una superficie che è garantito essere watertight e che non presenta altre imperfezioni quali t-junction o strutture non manifold: tuttavia, come abbiamo già sottolineato in 2.2.3, tale algoritmo è più orientato alla riparazione di mesh che non alla loro creazione per fusione di range map. La rappresentazione implicita basata su radial basis function consente di generare superfici continue ad un arbitrario livello di dettaglio, solo che il costo computazione necessario al fitting dell'RBF risulta estremamente alto, e quindi ampiamente in contrasto con gli obiettivi che ci eravamo prefissati. Per quanto riguarda infine gli algoritmi zero-set, MPU ed point set, le impressioni sono state decisamente positive in quanto tutti e tre gli algoritmi a nostro giudizio consentirebbero di definire delle superfici di qualità equivalente con un costo computazione ragionevole. Per tale motivo, l'algoritmo da noi implementato è basato su un approccio che è strettamente connesso sia ai metodi point-set sia quelli zero-set. Il motivo per cui abbiamo preferito questi ultimi algoritmi rispetto al metodo MPU è legato principalmente alla non completa chiarezza di vari parametri: infatti la sua descrizione così come fornita in [25] è espressa in termini di vari parametri, alcuni dei quali a nostro avviso mancano di una adeguata spiegazione.

Capitolo 3

Il framework

Come abbiamo visto nel capitolo precedente, molti degli algoritmi di integrazione basati su rappresentazioni implicite lavorano localmente su insiemi di punti spazialmente contigui. A questo scopo, considerando le caratteristiche del dataset in ingresso (vedi 3.1), abbiamo cercato di progettare una struttura dati tale da adattarsi a queste esigenze e che potesse essere usata anche per metodi di ricostruzione differenti.

L'algoritmo di fusione implementato è basato sui metodi point-set a cui si è accennato in 2.2.5 e pertanto assume che l'input sia costituito da una nuvola di punti dotati di normale, eventualmente arricchita da altre informazioni aggiuntive quali qualità e colore: prima di procedere alla sua descrizione, riteniamo sia utile fornire i dettagli implementativi anche del layer sottostante responsabile della gestione del dataset. Questo layer è stato sviluppato cercando di rispettare la stessa filosofia alla base della libreria "Visualization and Computer Graphics Lib" cosicché da essere ad essa integrabile: tale libreria, sviluppata ed ampiamente impiegata all'interno del gruppo di ricerca VCG, è una libreria C++ cross-platform templatata orientata alla gestione ed alla visualizzazione con OpenGL di mesh simpliciali e tetraedrali. L'integrazione alla libreria e la conseguente adozione dei principi alle sue fondamenta hanno guidato la realizzazione del layer di gestione del dataset verso la maggior generalità possibile, consentendo la creazione di un vero e proprio

framework per l'implementazione di vari algoritmi di estrazione di superfici. Nelle prossime sezioni si descriverà come all'interno di questo framework viene implementata l'organizzazione della nuvola di punti e come tale struttura venga poi sfruttata per l'efficiente risoluzione delle query spaziali.

3.1 L'input

Ogni scansione effettuata ad un oggetto durante la fase di acquisizione genera una range map all'interno della quale tutti i campioni sono espressi nel sistema di riferimento dello scanner al momento della ripresa: di conseguenza, ogni range map è espressa in un proprio sistema di riferimento differente da quello delle altre range map. Durante la successiva fase di allineamento, eventualmente appoggiandosi ai log dei sistemi di tracking dello scanner, ad ogni range map è associata una trasformazione rigida, cioè una coppia rotazione e traslazione, attraverso la quale è possibile esprimere le primitive in essa contenute in un sistema di riferimento comune a tutte le altre range map: l'unione quindi dei punti di tutte le range map espresse nel sistema di riferimento comune attraverso le varie trasformazioni rigide permette di definire infine la nuvola di punti. L'obiettivo di svincolare quanto più possibile l'algoritmo di integrazione dal formato con cui il dataset è rappresentato alla fine della fase di allineamento, ha suggerito di delegare interamente al framework la gestione dell'input; questa scelta ci ha consentito tra l'altro di impiegare l'algoritmo di integrazione sviluppato anche su dataset non provenienti dal processo di acquisizione, ma ottenuti attraverso conversione a nuvola di punti di modelli tridimensionali già esistenti. In entrambi i casi, ciò che viene passato in ingresso all'algoritmo di fusione è, indipendentemente da come il dataset sia stato generato, una nuvola di punti arricchita da informazioni ausiliari quali normali, colore e qualità. Il raggiungimento di questo livello di astrazione è stato possibile grazie alla particolare rappresentazione del da-

taset adottata all'interno del framework, vale a dire come una sequenza di *handle* a insiemi di punti: tali handle costituiscono l'unico punto d'accesso ai punti e, come sarà illustrato nella successiva sezione 3.4, giocano un ruolo fondamentale nell'estensione out-of-core. Il numero di handle generate per un dato input dipendono dalla tipologia di quest'ultimo. Se l'input è infatti un modello tridimensionale, l'insieme dei suoi vertici vengono interpretati come la nuvola di punti e vengono caricati all'interno di una sola handle; se invece l'input è un file di allineamento, per ogni range map in esso specificata viene generata una handle attraverso cui sarà poi possibile accedere ai punti in essa contenuti. In quest'ultimo caso, ogni handle provvede anche a portare la rispettiva range map nel sistema di riferimento comune: ai punti in essa contenuti viene applicata la trasformazione rigida, definita nel file di allineamento per la rispettiva range map, che permette di esprimere quest'ultima nel sistema di riferimento globale.

Indipendentemente dall'input, dal punto di vista del framework il dataset dunque non è altro che un insieme più o meno numeroso di handle all'interno delle quali è garantito che siano definiti per ogni punto anche normale, colore e qualità. Tali informazioni, nel caso in cui l'input sia un file di allineamento, sono già specificate per ogni punto all'interno delle varie range map, mentre se l'input è un modello tridimensionale tale informazioni possono essere ricavate dall'analisi del modello in fase di creazione della handle: la normale da associare ad un punto può infatti essere calcolata come media normalizzata delle normali alle facce che condividono un dato vertice mentre il colore ricavato dall'eventuale presenza delle texture. Facciamo notare che per i punti ricavati da un modello tridimensionale non è significativo parlare di qualità: quest'ultima ha infatti un senso se associata ai campioni presenti nelle range map, dove rappresenta una stima dell'errore di distorsione introdotto durante la fase di acquisizione, e quindi può essere interpretata come la confidenza dei campioni; in un modello tridimensionale invece tutti i vertici

vengono utilizzati per definire qualche porzione di superficie e di conseguenza risulterebbe irragionevole definire una stima della confidenza dei vertici: in questo caso la qualità è impostata ad un prefissato valore costante.

3.2 Spatial subdivision

3.2.1 Octree e octree coordinate

Il nucleo dell'intero framework è costituito da un *octree*, una struttura dati gerarchica largamente impiegata per la suddivisione e la gestione di dataset volumetrici. Dal punto di vista algoritmo un octree è un albero non necessariamente completo in cui ogni nodo ha al più otto figli. Un mapping fra nodi dell'albero e porzioni del volume consente quindi di rappresentare e di gestire lo spazio occupato dal dataset attraverso l'octree stesso: tale mapping associa alla radice dell'octree la bounding-box contenente l'intero dataset, e ad ogni altro nodo una porzione di spazio ottenuta suddividendo il volume associato al nodo padre. Le celle generate durante il processo di partizionamento del volume vengono chiamate *voxel* ed hanno dimensione decrescente all'aumentare della profondità del nodo cui sono associate. Il partizionamento del volume avviene attraverso piani paralleli agli assi e, ad ogni livello, la porzione di volume associata ad un nodo viene suddivisa in otto ottanti, tutti di pari dimensione, ognuno dei quali viene associato agli eventuali figli del nodo; in figura 3.1 è illustrato il partizionamento spaziale indotto da un quadtree, l'equivalente dell'octree in uno spazio bidimensionale.

Il processo di suddivisione si applica a partire dalla radice ricorsivamente sui suoi figli, fino a quando non viene più verificato un determinato predicato. Variazioni nella formulazione di tale predicato si ripercuotono ovviamente sulle proprietà esibite dall'octree: per esempio, se il predicato viene formulato in termini del numero di punti contenuti all'interno dei singoli voxel, l'octree

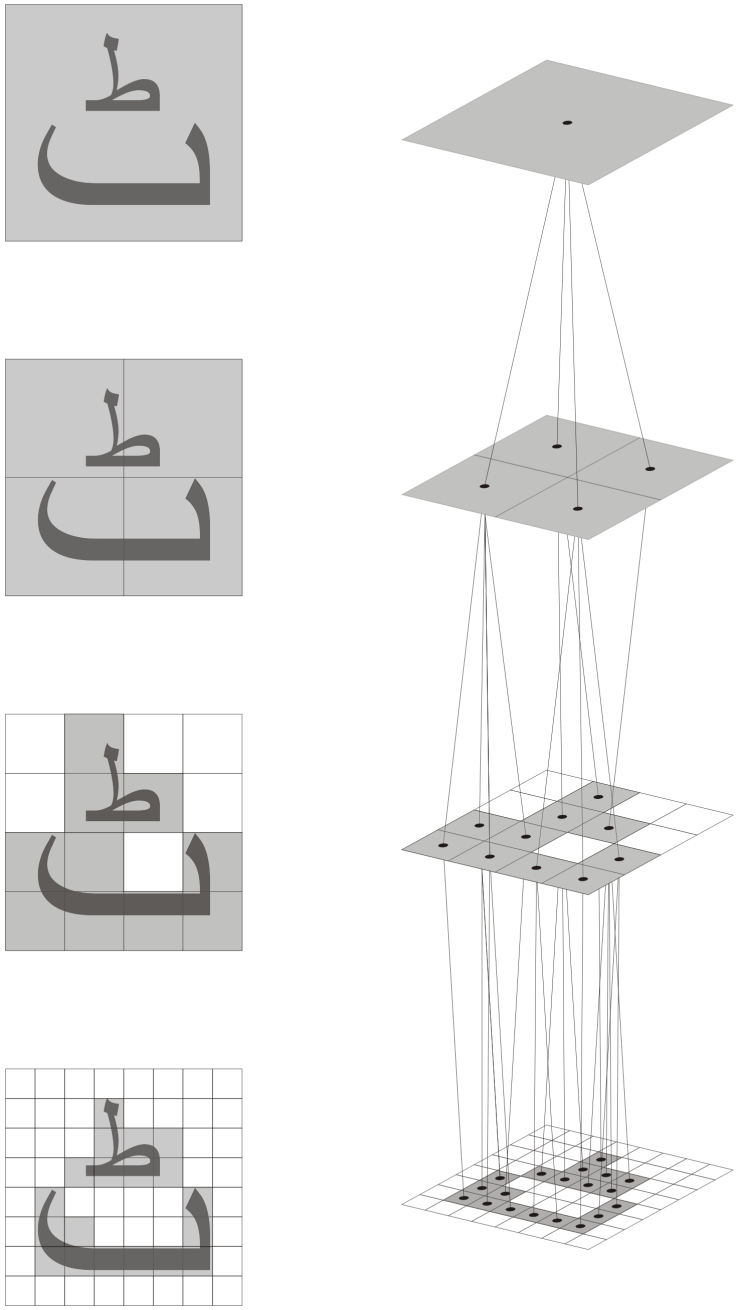


Figura 3.1: *Partizionamento di uno spazio bidimensionale con un quadtree. A sinistra, partizionamento di risoluzione crescente dall'alto verso il basso della porzione del piano occupata da una lettera dell'alfabeto arabo. A destra, la gerarchia fra i nodi del quadtree. I nodi effettivamente presenti nell'albero sono solamente quelli colorati.*

costruito risulta essere adattivo, nel senso che la sua profondità varia localmente in funzione della densità dei dati; se invece il predicato viene formulato in termini di lunghezza del cammino dalla radice fino al nodo cui un voxel è associato, allora l'octree risultante avrà tutte le sue foglie dislocate alla stessa profondità, indipendentemente dalla distribuzione dei dati. È chiaro che la scelta del tipo di predicato da utilizzare vada attentamente ponderata e di volta in volta rapportata alle esigenze degli algoritmi che utilizzeranno poi la struttura dati risultante: poiché l'obiettivo del nostro lavoro era l'implementazione di un efficiente algoritmo di estrazione di superfici, si è optato per una politica adattiva di suddivisione del volume basata sul numero k di campioni contenuti all'interno di un nodo, dove k è un parametro eventualmente impostato dall'utente. Questa scelta si è rivelata estremamente vantaggiosa in quanto la suddivisione adattiva del volume da un lato garantisce che la complessità della rappresentazione si adatti alla complessità del modello da ricostruire, mentre dall'altro consente un notevole risparmio di memoria, derivante dal pruning di quei sottoalberi responsabili della gestione di quelle porzioni di volume con densità di campionamento non sufficiente.

Il processo di costruzione dell'albero si appoggia ad un mapping tra nodi dell'albero e porzioni di volume la cui definizione è induttiva rispetto alla gerarchia dei nodi dell'octree; di conseguenza l'intero mapping risulta definito non appena venga imposta una valida corrispondenza per la radice. Poiché il mapping deve associare alla radice un volume pari almeno alla porzione di spazio occupato dall'intero dataset, per la sua corretta definizione è sufficiente imporre che il voxel corrispondente alla radice sia il minimo voxel cubico comprendente la bounding-box al cui interno risulti incluso l'intero dataset. Tale bounding box può essere calcolata parallelamente alla creazione delle varie handle: nel caso più semplice in cui l'input è un modello tridimensionale, tale bounding box coincide con la bounding box del modello stesso; nel caso invece che l'input sia un file di allineamento, la bounding box globale

per l'intero dataset viene determinata a partire dalle bounding box definite per le singole range map espresse nel sistema di riferimento comune. Queste bounding box vengono progressivamente calcolate dalle varie handle durante l'applicazione all'insieme dei punti contenuti nella propria range map della trasformazione rigida a questa associata durante la fase di allineamento: queste bounding box rappresentano un limite alla porzione di volume nel sistema di riferimento finale occupato dalle singole range map, e dunque la loro unione coincide con la porzione di volume occupata dall'intero dataset nel sistema di riferimento comune.

L'assegnamento della bounding box globale così definita alla radice dell'octree e la definizione ricorsiva del mapping consentono di associare in modo univoco una porzione di volume ad ogni nodo dell'albero. Sebbene il calcolo del volume associato ad ogni nodo sia immediato una volta noti il voxel assegnato alla radice ed il cammino che da questa porta fino al nodo, è preferibile tenere traccia del mapping ad ogni nodo dell'albero. Ciò che viene mantenuto ad ogni nodo non è, diversamente da quanto una prima valutazione suggerisce, il voxel associato dal mapping al nodo stesso, ma bensì l'insieme minimo di informazioni per l'identificazione del voxel, vale a dire la profondità del nodo ed il centro del voxel espresso in *octree coordinate*. A spingere verso questa scelta è stato il vincolo che l'overhead in termini di occupazione di memoria dovuto dalla struttura dati fosse il minimo possibile: memorizzare infatti ad ogni nodo il voxel stesso espresso in *world coordinate* avrebbe richiesto per ogni nodo ben 25 byte, che è lo spazio necessario a memorizzare i due punti estremi della diagonale del voxel espressi in virgola mobile ($2 \times (3$ `float`$)$) più la profondità del nodo (1 `char`); tra l'altro l'adozione di questa scelta avrebbe comportato un'inutile replicazione dell'informazione, in quanto l'estremo di ogni diagonale sarebbe stato ripetuto in almeno due nodi. Grazie alla rappresentazione da noi adottata, l'overhead ad ogni nodo per la rappresentazione del voxel è solamente di 7 byte, che è lo spazio necessario a

memorizzare la profondità del nodo (1 `char`) più il centro del voxel espresso in octree coordinate (3 `short`). Le octree coordinate altro non sono che un modo compatto di rappresentare una qualche posizione spaziale associata ad un nodo in funzione dalla sua profondità e dell'altezza dell'albero. La loro definizione viene suggerita della regola di partizionamento del volume adottata nella costruzione dell'octree, in base alla quale il voxel associato ad ogni nodo viene partizionato da tre piani paralleli agli assi in otto ottanti di pari dimensioni; di conseguenza, il volume del voxel associato ad un nodo è $\frac{1}{8}$ del volume del voxel associato al padre, il cui significato è che la risoluzione ad ogni livello aumenta di un fattore 2 lungo ogni dimensione. Se dunque, come nel nostro caso, è nota la massima altezza h dell'albero, allora è nota anche la massima risoluzione che verrà raggiunta alle sue foglie, che è 2^h. Tuttavia, l'intervallo dei valori interi $[0, \ldots, 2^h - 1]$ è sufficiente all'assegnamento di un indice ad ogni voxel, ma non alla codifica del loro centro. Notando però che il centro di un voxel associato ad un nodo altro non è che uno dei vertici dei voxel associati ai figli di quel nodo, risulta immediato considerare lo spazio di coordinate intere $[0, \ldots, 2^{h+1} - 1]$ che sarebbe necessario per generare gli indici dei voxel per un albero di altezza $h + 1$ (figura 3.2). La generazione delle coordinate dei centri dei voxel è ricorsiva ed ha inizio dalla radice dell'octree; per tutti gli altri nodi, il centro del voxel loro associato viene definito in funzione del centro del voxel associato al nodo padre e del proprio indice di figlio. Nella nostra implementazione il centro del voxel assegnato alla radice ha tutte le coordinate pari alla risoluzione massima che viene raggiunta alle foglie, vale a dire 2^h, con h che rappresenta la massima altezza dell'albero; per ogni altro nodo n, k-esimo figlio del nodo $f = \texttt{father}(n)$, con $k \in [0, \ldots, 7]$, la procedura che definisce il centro del voxel associatogli è:

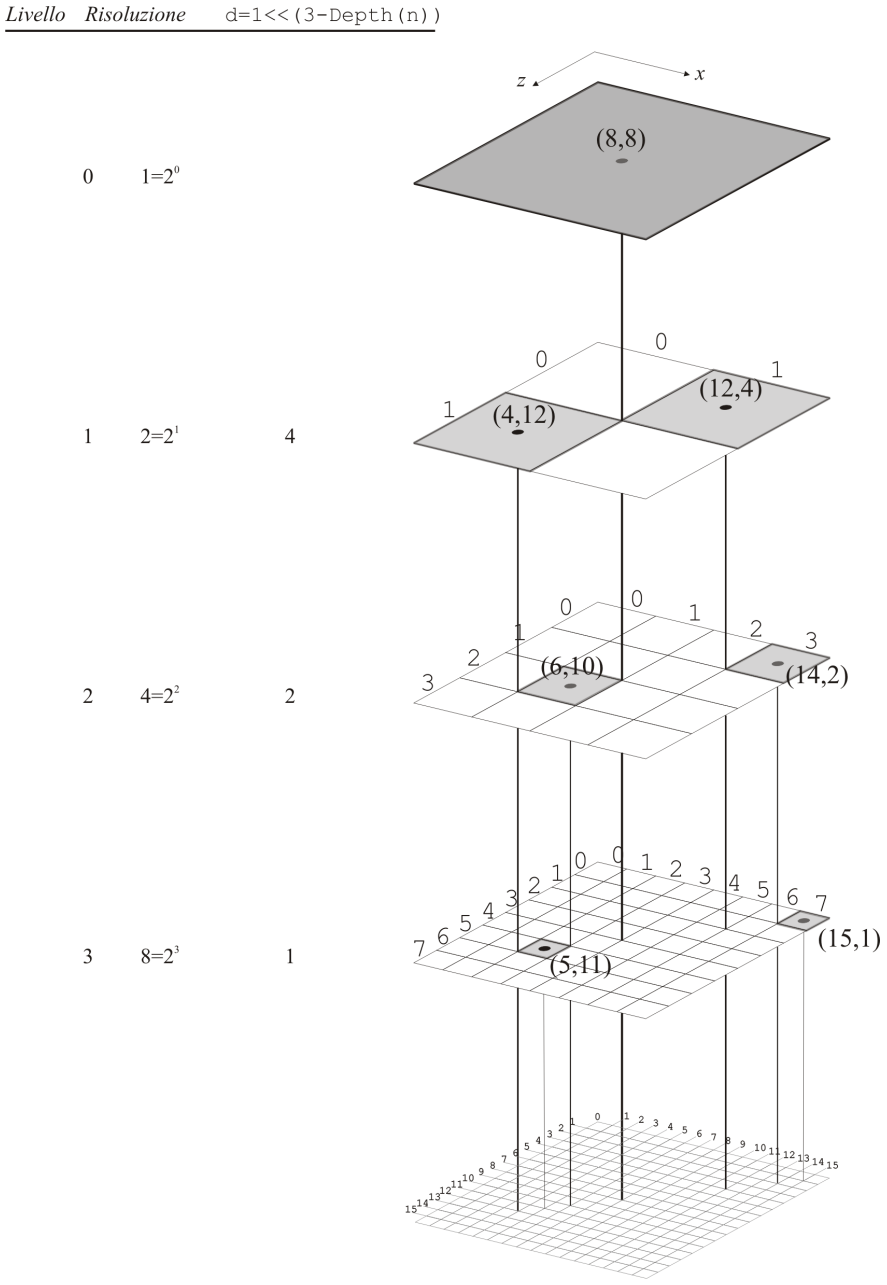


Figura 3.2: *Definizione dei centri in octree coordinate associati ai voxel di un octree di profondità massima 3. Com'è possibile osservare le coordinate di un ogni nodo ad una qualsiasi altezza vengono generate attraverso opportune addizioni/sottrazione della quantità d rispetto alle coordinate del voxel associato al padre e possono essere messe in relazione con gli indici di una griglia di risoluzione* 2^{3+1}.

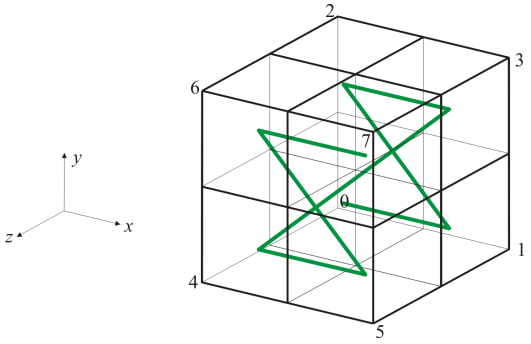


Figura 3.3: *L'ordinamento fra gli ottanti generati dal partizionamento spaziale di un voxel è coerente con la curva space-filling z-order.*

```
int d = 1<<(h-Depth(n));
Center(n).X = Center(f).X + ((k&1)? d : -d);
Center(n).Y = Center(f).Y + ((k&2)? d : -d);
Center(n).Z = Center(f).Z + ((k&4)? d : -d);
```

in cui gli operatori `<<` e `&` denotano rispettivamente il bitwise-left-shift ed il bitwise-and standard del C++. In tale procedura è implicito un ordinamento fra i figli di ogni nodo (vedi figura 3.3): tale ordinamento è detto *z-ordering* e costituisce il metodo alla base della procedura di indicizzazione del dataset. Risalire alla porzione di spazio in world coordinate associato ad ogni nodo è a questo punto immediato, in quanto si riduce a calcolare un cambiamento del sistema di riferimento: a partire dal nodo n, il voxel associatogli viene univocamente identificato dalla procedura:

```
Point3f node_dim = bb.Dim()/(1<<depth(n));
int shift = (h-Depth(n))+1;
Voxel.min = bb.min+(node_dim*(Center(n)>>shift));
Voxel.max = Voxel.min+node_dim;
```

in cui `bb` indica la bounding-box che racchiude il volume occupato dall'intero dataset (cioè il voxel associato alla radice) e `Center(n)>>shift` è una

notazione più compatta per indicare l'applicazione dell'operatore bitwise-left-shift ad ogni componente del punto `Center(n)`. È evidente che il mapping tra nodi dell'octree e porzione di volume non viene mantenuto esplicitamente, ma viene ogni volta calcolato attraverso la precedente procedura a partire dalla informazioni presenti nei nodi dell'albero: a farci optare per questa scelta è stata la considerazione che la sua completa memorizzazione avrebbe richiesto un quantitativo di memoria almeno proporzionale alla dimensione dell'albero, mentre l'overhead dovuto al suo calcolo si è rivelato decisamente trascurabile.

3.2.2 Z-Ordering

Lo z-ordering [26, 27] è un'efficiente tecnica basata sull'indicizzazione della curva di Lebesgue che permette di mappare dati appartenenti ad uno spazio n-dimensione in una sequenza monodimensionale. Il suo utilizzo si rivela estremamente vantaggioso sia perché la sequenza generata può essere raggruppata gerarchicamente per livelli di risoluzione, sia perché tale ordinamento può essere calcolato in modo molto efficiente esclusivamente attraverso la manipolazione dei bit che rappresentano il dato n-dimensionale. La definizione dello schema gerarchico di indicizzazione deriva dal sottocampionamento di una griglia effettuato seguendo la curva space-filling di Lebesgue.

Si consideri un insieme $\mathscr{P}$ di n elementi decomposti in una gerarchia $\mathscr{H}$ di k livelli di risoluzione $\mathscr{H} = \{\mathscr{P}_0, \mathscr{P}_1, \ldots, \mathscr{P}_{k-1}\}$ tale che:

$$\mathscr{P}_0 \subset \mathscr{P}_1 \subset \ldots \subset \mathscr{P}_{k-1} = \mathscr{P},$$

in cui l'i-esimo livello di risoluzione $\mathscr{P}_i$ è detto *meno raffinato* del livello j-esimo $\mathscr{P}_j$ se $i < j$. L'ordinamento degli elementi in $\mathscr{P}$ è definito da una funzione di cardinalità $I : \mathscr{P} \to \{0, \ldots, n-1\}$, che verifica l'identità

$$\mathscr{P}[I(p)] \equiv p,$$

dove le parentesi quadre denotano l'accesso per indice agli elementi dell'insieme. A partire da $\mathscr{P}$ ed $\mathscr{H}$, è possibile derivare una nuova sequenza di insiemi $\mathscr{P}'$ definiti come:

$$\mathscr{P}'_i = \mathscr{P}_i \backslash \mathscr{P}_{i-1} \qquad i = 0, \ldots, k-1,$$

in cui formalmente $\mathscr{P}_{-1} = \emptyset$, attraverso i quali generare una nuova sequenza $\mathscr{H}' = \{\mathscr{P}'_0, \mathscr{P}'_1, \ldots, \mathscr{P}'_{k-1}\}$ che sia un partizionamento di $\mathscr{P}$. Similmente, è possibile derivare anche una nuova funzione di cardinalità $I' : \mathscr{P} \to \{0, \ldots, n-1\}$ sulla base delle seguenti due proprietà:

- $\forall\, p, q \in \mathscr{P}'_i : I(p) < I(q) \iff I'(p) < I'(q)$
- $\forall\, p \in \mathscr{P}'_i,\ \forall\, q \in \mathscr{P}'_j : i < j \Rightarrow I'(p) < I'(q)$.

Se la funzione di partenza I, quando ristretta ad un qualsivoglia livello di risoluzione $\mathscr{P}_i$, esibisce una forte località, allora attraverso la funzione di cardinalità derivata I' è possibile generare una indicizzazione globale che sia consistente fra i diversi livelli della gerarchia. La nuova funzione può quindi essere individuata in due fasi, prima determinando il numero di elementi all'interno di ogni insieme derivato $\mathscr{P}'_i$, quindi definendo una nuova funzione di cardinalità $I''_i = I'|_{\mathscr{P}'_j}$ ottenuta come restrizione di I' ad ogni insieme $\mathscr{P}'_j$. Se c_i è il numero di elementi di $\mathscr{P}'_i$, allora è possibile determinare un indice iniziale per gli elementi in un dato livello di risoluzione calcolando la sequenza $C_0, \ldots, C_{k-1}$ come

$$C_i = \sum_{j=0}^{i-1} c_j.$$

A partire da queste costanti è possibile definire l'insieme delle funzioni di cardinalità $I''_i : \mathscr{P}'_j \to \{0, \ldots, c_i - 1\}$ attraverso le quali definire I', cioè:

$$\forall\, p \in \mathscr{P}'_i : I'(p) = C_i + I''_i(p).$$

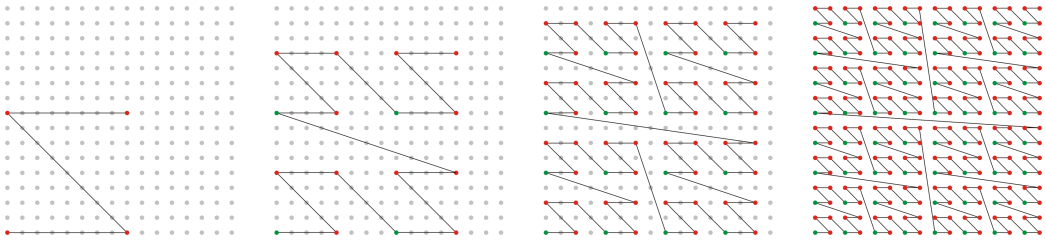


Figura 3.4: *La sequenza delle curve space-filling di Lebesgue utilizzata per definire una struttura gerarchica su una griglia bidimensionale. Tra due livelli consecutivi della gerarchia, i punti in verde rappresentano i dati* meno raffinati*, mentre i vertici in rosso indicano l'informazione al livello inferiore di risoluzione.*

Poiché l'individuazione delle costanti C_i può essere effettuato durante la fase di preprocessing, a run-time il calcolo di $I'(p)$ si riduce alla somma fra la costante C_i, nota non appena sia individuato il livello di risoluzione i di s, ed il valore $I''_i(p)$. Tali funzioni I''_i di cardinalità locale possono essere derivate dalla curva di Lebesgue per gerarchie di alberi 2^n-ari.

La curva space-filling di Lebesgue viene anche chiamata, per via della sua forma nello spazio bidimensionale (figura 3.4), curva space-filling Z-order. La sua definizione in questo spazio viene espressa induttivamente a partire da una Z-shape per il primo livello di risoluzione di dimensione 1: l'i-esimo livello di risoluzione è definito come la curva ottenuta sostituendo i vertici della curva al livello $(i-1)$ con una Z-shape di dimensione $\frac{1}{2^i}$. La struttura gerarchica della curva space-filling per il caso tridimensionale rimane invariata, mentre cambia la forma della Z-shape iniziale, che viene sostituita da una coppia di Z-shape connesse giacenti su due facce opposte di un cubo. Similmente, la versione d-dimensionale della curva space-filling ha anch'essa stessa struttura gerarchica mentre la sua forma iniziale è definita come coppia connesse di Z-shape base $(d-1)$-dimensionali disposte sulle facce opposte

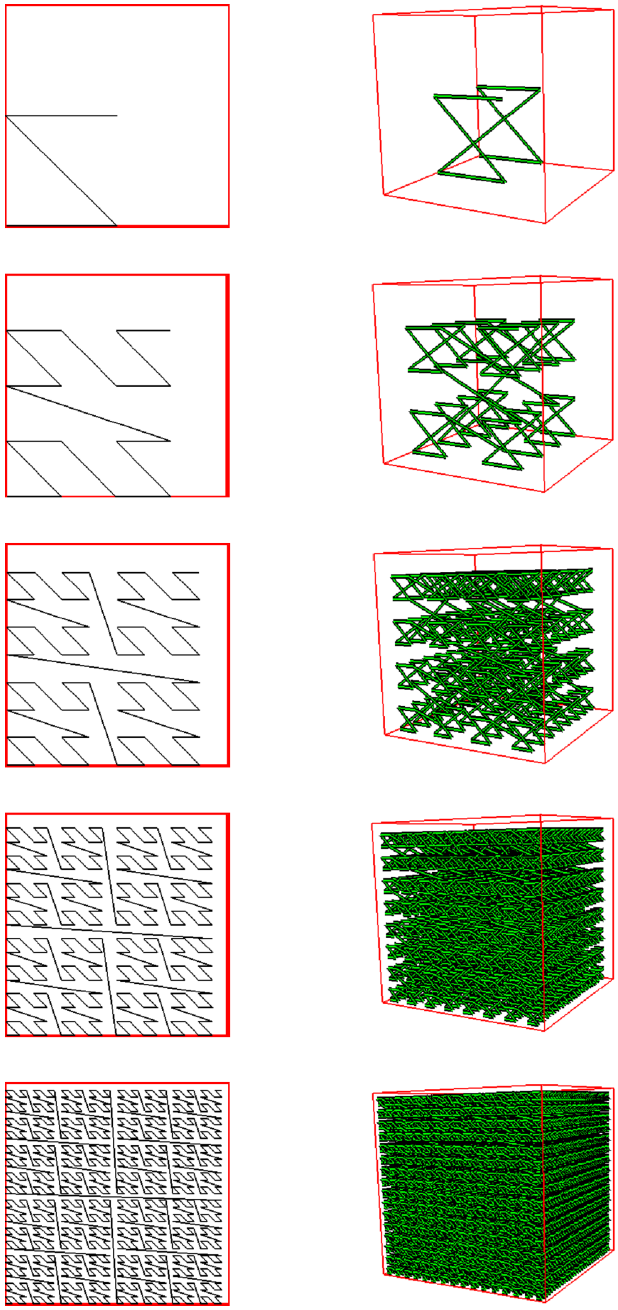


Figura 3.5: *I primi cinque livelli di risoluzione delle curva space-filling di Lebesgue in uno spazio bidimensionale (a sinistra) e in uno spazio tridimensionale (a destra).*

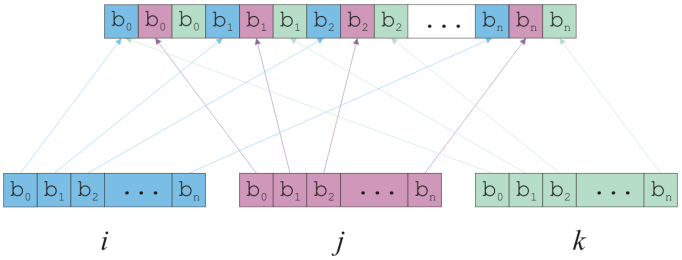


Figura 3.6: *Costruzione dell'indice monodimensionale per la curva space-filling di Lebesgue. Nel caso tridimensionale, l'indice originale è un insieme di tre stringhe di bit (i, j, k). L'indice monodimensionale I è ottenuto attraverso il mescolamento dei bit della rappresentazione binaria di i, j e k in una singola stringa di bit.*

di un cubo d-dimensionale. In figura 3.5 è illustrata la forma della Z-shape per il caso bidimensionale e quello tridimensionale.

Da un punto di vista più pratico è interessante notare come, fissato un arbitrario livello di risoluzione, se si immerge una curva space-filling di Lebesgue in una matrice d-dimensionale, la sua sagoma attraversa tutte le celle della matrice secondo un ben prestabilito ordine. In altre parole seguendo l'ordine con cui la curva space-filling Z-order attraversa le celle della matrice, è possibile mappare gli indici d-dimensionali che identificano le celle in una sequenza monodimensionale ordinata di valori. Se una cella e della matrice è individuata dagli indici $(i_1, \ldots, i_d)$, il valore associatole nella sequenza monodimensionale viene calcolato attraverso un opportuno riordinamento dei bit che costituiscono la rappresentazione binaria degli indici $i_1, \ldots, i_d$: più precisamente, se la rappresentazione binaria dell'indice i_j, per $j \in [1, \ldots, d]$, è la stringa di h bit $<\mathtt{b}_j^1\mathtt{b}_j^2 \ldots \mathtt{b}_j^h>$, allora il valore che rappresenta e nella sequenza monodimensionale è codificato, come illustrato in figura 3.6, con la stringa di hd bit $<\mathtt{b}_1^1\mathtt{b}_2^1 \ldots \mathtt{b}_d^1\mathtt{b}_1^2\mathtt{b}_2^2 \ldots \mathtt{b}_d^2\mathtt{b}_1^h\mathtt{b}_2^h \ldots \mathtt{b}_d^h>$.

L'ordinamento che la curva space-filling di Lebesgue impone ad una gri-

glia d-dimensionale per un fissato livello di risoluzione k, può essere organizzato in una struttura gerarchica ad albero di altezza k: a tal proposito basti osservare che ogni griglia d-dimensionale può essere descritta per mezzo di un opportuno mapping attraverso un albero 2^d-ario. Per esempio, è possibile descrivere una griglia bidimensionale per mezzo di un quadtree (un albero quaternario), o, come abbiamo fatto in precedenza, una griglia tridimensionale per mezzo di un octree (un albero ottario). Questa correlazione tra la curva Z-order e la struttura degli alberi 2^d-ari ci permette inoltre di individuare facilmente i tre elementi necessari alla definizione della funzione cardinalità I' prima definita, vale a dire:

- il livello di un elemento i: se l'albero 2^n-ario ha altezza k, allora l'elemento che occupa la j-esima posizione dello Z-order appartiene al livello $k - h$ della gerarchia, dove h è il numero di trailing-zero nella rappresentazione binaria del rapporto fra j e d, cioè

$$i = k - \texttt{RightmostZeros}\left(\frac{j}{d}\right)_2$$

- le costanti C_i: il numero di elementi presenti all'interno della gerarchia nell'insieme dei livelli meno raffinati di i è:

$$C_i = \begin{cases} 2^{d(i-1)} & i > 0 \\ 0 & i = 0 \end{cases}$$

- la funzione di cardinalità locale I''_i: se il valore di un elemento appartenente all'insieme $\mathscr{P}''_i$ nella sequenza monodimensionale è j, allora uno dei suoi ultimi d bit è diverso da zero; il valore associatogli dalla funzione di cardinalità locale è allora:

$$I''_i(j) = \left\lfloor \frac{j}{2^{di}} \right\rfloor - \left\lfloor \frac{j}{2^{d(i+1)}} \right\rfloor - 1.$$

3.2.3 Indicizzazione del dataset

La corrispondenza definita dal mapping tra nodi dell'albero e porzioni del volume costituisce il punto di partenza del processo di indicizzazione del dataset: infatti per mezzo di questa corrispondenza ogni punto appartenente al dataset può essere associato ad un insieme ben particolare di nodi, quelli cioè all'interno del cui voxel risulti compreso. L'idea alla base dell'indicizzazione del dataset è cioè quella di tenere traccia, in ogni nodo dell'octree, dell'insieme dei punti che cadono all'interno del voxel associatogli dal mapping. Poiché questa informazione va mantenuta ad ogni nodo, la scelta di come rappresentarla è un punto cruciale per la realizzazione di una struttura dati robusta e scalabile. Sicuramente la sua rappresentazione dovrebbe essere la più compatta possibile, in quanto si vorrebbe evitare che il costo dell'indicizzazione costituisse una considerevole percentuale del costo dell'intera struttura dati: al riguardo è utile far notare che associare un punto ad una foglia dell'albero implica doverlo associare anche a tutti i suoi antenati fino alla radice, in quanto il voxel associato alla foglia risulta per costruzione incluso nei voxel associati a tutti i suoi antenati. Inoltre, tale rappresentazione deve essere scelta tenendo sempre presente l'estensione out-of-core, cosicché l'accesso ai punti possa essere realizzato in maniera uniforme, indipendentemente dalla loro effettiva presenza in memoria principale. La soluzione da noi delineata è formulata sulla base di come le range map sono memorizzate e rappresentate all'interno dalle handle introdotte in precedenza: tali handle vengono utilizzate non solo in fase di caricamento delle range map, ma costituiscono all'interno del framework l'unico modo per accedere ai punti del dataset. In particolare, dopo il caricamento, la trasformazione e l'arricchimento della range map con le informazioni deducibili dalla sua analisi, i punti in essa contenuti vengono ordinati e memorizzati sfruttando la tecnica dello z-ordering. La chiave di ordinamento assegnata ad ogni punto coincide con

lo z-order del voxel di dimensione minima all'interno di cui risulti compreso: tale voxel viene individuato effettuando una visita top-down dell'octree, a partire dalla radice fino a quando non viene raggiunto un nodo per cui non è più verificato il predicato che ne causa la suddivisione. Tale visita è ricorsiva e ad ogni nodo la scelta del figlio su cui proseguire viene effettuata confrontando le coordinate spaziali del punto con le world coordinate dei voxel associati ai figli del nodo: per costruzione dei voxel infatti, ad ogni passo il punto risulerà compreso solamente in uno degli otto voxel associati ai figli del nodo corrente; di conseguenza, per ogni punto nella range map, esisterà un solo valido cammino dalla radice fino ad una qualche foglia. Il motivo per cui la chiave di ordinamento associata ad ogni punto è lo z-order del voxel all'interno di cui cade, è legato al discorso delle gerarchie di risoluzione affrontato in 3.2.2: ordinare i punti rispetto alla risoluzione più fine implica l'ordinamento rispetto ad una qualsiasi altra risoluzione all'interno della gerarchia. Questa scelta si riflette anche sul processo di indicizzazione dell'intero octree, che avviene, come verrà illustrato successivamente, seguendo un approccio bottom-up.

Ovviamente, affinché i punti appartenenti ad una data range map possano essere ordinati, è necessario che per ognuno di essi sia stato definito lo z-order o, equivalentemente, sia stata individuata la foglia dell'octree all'interno del cui voxel risulti contenuto. A questo punto, tutte quelle foglie dell'albero all'interno del cui voxel ricada almeno un punto di tale range map, vengono aggiornate con le informazioni di indicizzazione relative a tale range map. L'informazione che viene inserita in ognuna di queste foglie è costituita da una coppia di valori `(id, range)`, dove `id` è un identificativo unico per mezzo del quale è possibile risalire alla handle che rappresenta questa range map, mentre `range` è a una coppia di indici del tipo `[begin, end)` che rappresenta il sottoinsieme dei punti di questa range map che cadono all'interno del voxel associato alla foglia. Tale rappresentazione è possibile

solo ed esclusivamente grazie all'ordinamento z-order imposto sui punti, in quanto tutti i punti che cadono all'interno del voxel associata ad una stessa foglia avranno necessariamente stessa chiave di ordinamento e, cioè, stesso z-order. Poiché in genere le range map risultano sovrapposte, nel senso che la stessa porzione di superficie è scandita più volte, di conseguenza, all'interno di ogni foglia possono cadere punti appartenenti a range map diverse, i vari riferimenti ai punti sono per ogni foglia organizzati all'interno di una lista. Ricostruire quindi il sottoinsieme dei punti del dataset che cadano all'interno del voxel associato ad una data foglia risulta immediato, in quanto si riduce semplicemente a scorrere la lista memorizzata all'interno della foglia.

3.2.4 Indicizzazione del dataset ai nodi interni

Non appena tutte le range map sono state processate e le foglie dell'octree conseguentemente arricchite con i riferimenti alla nuvola di punti, ha inizio il processo di indicizzazione dei nodi interni. Il motivo per cui tale fase è rimandata fino alla completa indicizzazione delle foglie è legato alla politica di costruzione dell'albero, che avviene, per evitarne la sua completa allocazione, parallelamente all'indicizzazione delle varie range map. Infatti, poiché non possono essere noti a priori né il numero di punti nel dataset né soprattutto la loro distribuzione nello spazio, la creazione dell'albero avviene insieme all'analisi dei dati, cosicché la sua espansione sia adattiva e coerente con la dislocazione dei punti. Più precisamente, quando durante la fase di ordinamento dei punti di una range map l'albero viene visitato per determinare lo z-order da assegnare ai punti, può capitare che non tutti i nodi siano presenti per completare il cammino dalla radice fino a qualche foglia: in questo caso è la stessa procedura che calcola lo z-order di un punto che provvede all'allocazione di tutti i nodi necessari per il raggiungimento di una foglia. L'albero costruito da questa procedura è quindi una struttura dati adattiva,

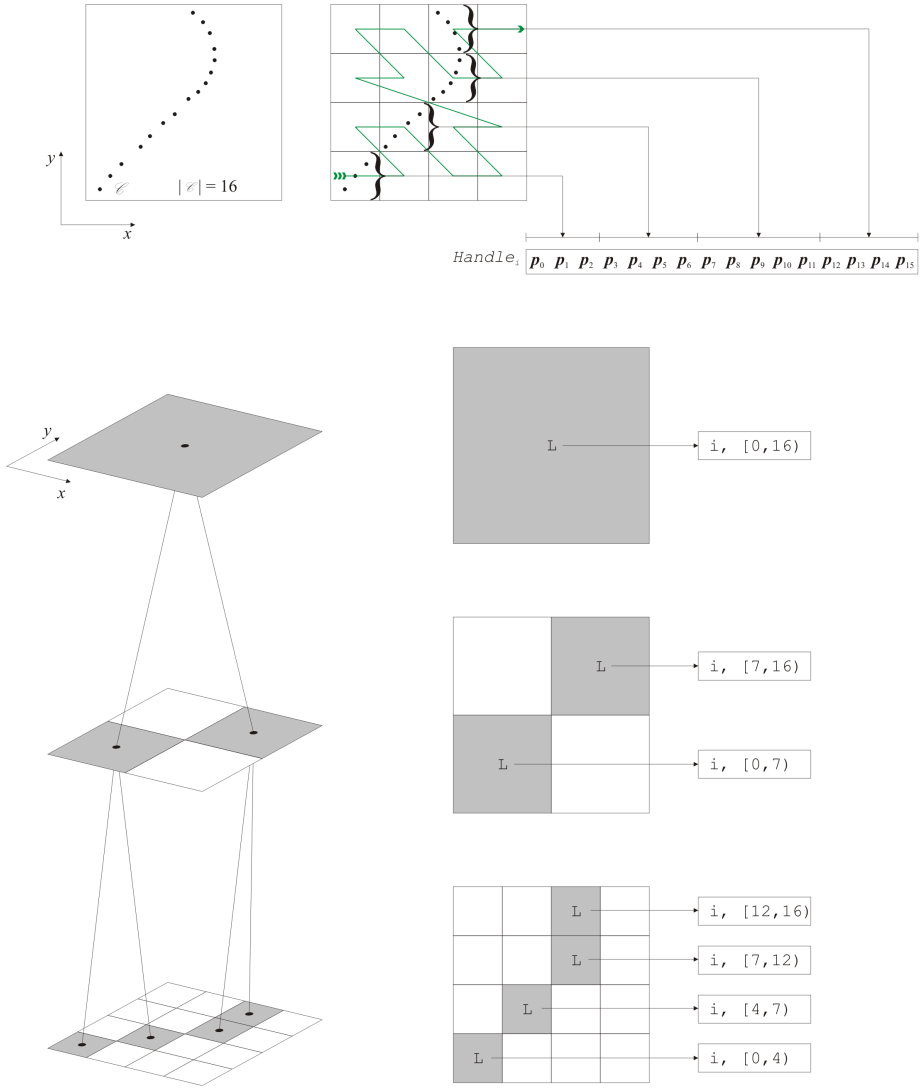


Figura 3.7: *Indicizzazione del dataset $\mathscr{C}$ nel caso bidimensionale per un quadtree di altezza 2. L'ordinamento dei punti imposto dalla curva z-order è illustrato in alto a destra. La lista* L *associata ad ogni nodo memorizza i riferimenti ai punti contenuti all'interno della porzione di volume associata al nodo stesso: tale insieme dei punti è codificato con un intervallo che, indipendentemente dalla profondità del nodo, risulta continuo e, per ogni nodo interno, risulta calcolabile dall'unione delle liste* L *associate ai suoi figli.*

in quanto la profondità massima viene raggiunta solamente per descrivere le porzioni del volume in cui cadano almeno k punti.

Quando tutte le range map sono state caricate, il processo di costruzione dell'octree può considerarsi terminato e può quindi essere completato il processo di indicizzazione dei nodi interni, che avviene per propagazione delle informazioni presenti nelle foglie. La formulazione del processo di indicizzazione dei nodi interni è anch'essa induttiva rispetto alla gerarchia dell'albero, solo che questa volta l'approccio seguito è di tipo bottom-up. L'indicizzazione dei nodi interni all'albero si basa sulla proprietà che lo z-ordering calcolato al livello più basso della gerarchia è consistente ad un qualsiasi altro livello non minore, e può essere spiegata ricorrendo ancora una volta alla politica di partizionamento spaziale impiegata in fase di costruzione dell'octree. Infatti i voxel associati ai figli di un nodo altro non sono che gli ottanti derivanti dal partizionamento del voxel associato al nodo stesso; di conseguenza, se un punto cade all'interno del voxel associato ad uno dei figli, per costruzione dei voxel questo punto cade anche all'interno del voxel associato al padre. Si consideri un nodo interno n dell'albero non ancora indicizzato i cui figli siano però già stati tutti visitati dalla procedura di indicizzazione: è evidente che la lista dei punti contenuti all'interno del voxel associato ad n può essere costruita semplicemente unendo le liste definite per ognuno dei suoi figli. Più precisamente, la lista dei punti contenuti all'interno del voxel associato al nodo n viene costruita iterando sulle liste associate ai figli: se all'interno di una di queste liste è presente un riferimento `(id, range)` alla range map rappresentata dalla handle `id` e nella lista associata ad n ancora non compare nessuna entry per la range map `id`, allora il riferimento `(id, range)` viene aggiunto alla lista del padre; viceversa, se nella lista associata ad n è già presente una entry per la handle `id`, allora l'intervallo `range` al padre viene modifico in modo tale da includere anche i punti presenti all'interno del voxel associato al figlio. Se l'ordine di visita dei figli è consistente con

lo z-order ed avviene cioè seguendo la numerazione riportata in figura 3.3, tale aggiornamento si riduce a modificare esclusivamente l'estremo superiore dell'intervallo `[begin, end)` al nodo padre, che va impostato al valore `range.end` specificato nel nodo figlio. In figura 3.7 è illustrato il processo di indicizzazione nel caso bidimensionale ai nodi di un quadtree. Al riguardo è interessante far notare che l'unione degli intervalli specificati ai nodi figli genera sempre un intervallo continuo al nodo padre, e che inoltre la lunghezza degli intervalli cresce progressivamente man mano che dalle foglie si risale fino alla radice, nella quale, l'intervallo specificato per ogni range map avrà estremo inferiore pari a zero ed estremo superiore pari al numero di punti compresi nella range map. Ad ogni nodo dell'octree quindi le informazioni memorizzate per l'indicizzazione del dataset è sempre una coppia di interi e dunque risulta indipendente dal numero di punti effettivamente contenuti all'interno della porzione di volume associatagli dal mapping; ovviamente, tale rappresentazione cresce linearmente con il numero delle range map necessarie alla completa descrizione di un oggetto.

3.3 Spatial query

Nell'ambito della ricostruzione di superfici un problema indistintamente comune a tutti i metodi è quello di rintracciare nel modo più efficiente e veloce possibile l'insieme dei punti che verifichino una certa proprietà spaziale: le interrogazioni più comuni riguardano l'insieme di punti dislocati in un circoscritto volume oppure l'insieme dei k punti più vicini ad un determinato altro punto. Il framework da noi implementato fornisce metodi ad alto livello per rispondere a simili query spaziali: la loro implementazione sfrutta quanto più possibile l'organizzazione spaziale definita in precedenza, grazie alla quale siamo riusciti a raggiungere tempi di esecuzione decisamente soddisfacenti.

3.3.1 Range query

Tra le varie interrogazioni spaziali quelle di più semplice risposta sono certamente le range query, in cui viene chiesto di individuare l'insieme di punti che cadono all'interno di un determinato volume; nella maggior parte dei casi, questo volume viene descritto attraverso o una bounding-box allineata agli assi o una sfera. In entrambi i casi, la risposta alla query viene generata attraverso una visita top-down dell'octree guidata dalle relazioni spaziali tra la primitiva specificata nella query ed i voxel associati ad ogni nodo. Descriveremo innanzitutto il caso più semplice in cui la primitiva della query è una bounding box allineata agli assi, e successivamente forniremo le modifiche da apportare a questa procedura per gestire il caso in cui la primitiva della query sia una sfera.

Prima di descrivere come all'interno del framework viene generata la risposta alla query, è utile far osservare che una bounding box allineata agli assi può essere rappresentata senza ambiguità attraverso due soli punti, quelli posti agli estremi di una delle sue diagonali. Poiché questa rappresentazione è valida anche per la descrizione di un voxel, tutti i test per determinare le relazioni di intersezione o inclusione fra una bounding box e un voxel possono essere implementati per mezzo di opportuni confronti fra le coppie di punti che li rappresentano; per esempio, il test di collisione fra una bounding box `bb` ed un voxel `v` è realizzato come:

```
1 bool Collide(Box &b, Voxel &v)
2 {
3     return (b.min.X<v.max.X && b.max.X>v.min.X
4          && b.min.Y<v.max.Y && b.max.Y>v.min.Y
5          && b.min.Z<v.max.Z && b.max.Z>v.min.Z);
6 };
```

mentre quello di inclusione come:

```
1 bool IsIn(Box &b, Voxel &v)
2 {
3     return (b.min.X<v.min.X && v.max.X<b.max.X
4           && b.min.Y<v.min.Y && v.max.Y<b.max.Y
5           && b.min.Z<v.min.Z && v.max.Z<b.max.Z);
6 };
```

Grazie a questi due metodi, l'insieme dei punti contenuti all'interno di una data bounding-box allineata agli assi può essere individuato per mezzo di una procedura di visita ricorsiva che ha origine nella radice dell'octree: ad ogni nodo la scelta dei figli da visitare è effettuata per mezzo di test di intersezione fra la bounding-box specificata nella query ed i voxel associati ad ognuno dei figli. Più precisamente, se il voxel `v` associato ad un dato figlio s non collide con la bounding box `bb` della query, allora l'intero sottoalbero radicato in s non verrà visitato dalla procedura; se invece il voxel `v` di s collide con la bounding-box `bb`, allora s viene marcato per la visita; se infine il voxel `v` risulta strettamente incluso in `bb`, allora tutti i punti del dataset inclusi in `v`, codificati nella lista associata ad s, vengono aggiunti all'insieme dei punti restituiti dalla query. Al fine di nascondere i dettagli di come il dataset venga mantenuto all'interno del framework, l'insieme dei punti individuati viene restituito al chiamante come array di puntatori ai punti memorizzati nelle handle.

La stessa struttura della procedura appena delinata può essere impiegata per individuare l'insieme dei punti racchiusi all'interno di una sfera a meno di sostituire il test di intersezione cubo-voxel con quello sfera-voxel. Tuttavia una tale implementazione si rivelerebbe decisamente inefficiente a causa del costo computazionale del test per l'individuazione di eventuali intersezioni fra una sfera ed un voxel: tale test richiederebbe infatti di verificare innanzitutto le relazioni spaziali fra la sfera e gli otto vertici del cubo, per ognuno dei quali risulta necessario il calcolo della distanza euclidea dal centro della sfera;

a seconda dei casi, questi semplici test non sono neppure sufficienti a rilevare l'intersezione fra il voxel e la sfera ed è dunque necessario effettuare ulteriori test di intersezione fra quest'ultima ed i piani su cui giacciono le facce del voxel: di conseguenza, il costo complessivo non risulta neanche paragonabile al test di intersezione cubo-voxel che, come si è illustrato in precedenza, si riduce al confronto di sole sei coordinate. Tuttavia l'osservazione che le query spaziali mirate ad individuare i punti all'interno di una data sfera sono circoscritte a porzioni di volume molto limitate ci suggerisce di formulare questa procedura completamente in termini di quella delineata in precedenza per bounding box allineate agli assi: più precisamente, i punti all'interno di una sfera possono essere individuati ricercando prima i punti racchiusi all'interno del cubo minimo al cui interno la sfera risulti inscritta, ed eliminando poi dall'insieme di punti così individuati quelli interni al cubo ma non alla sfera. Più precisamente, data la sfera $b = (\boldsymbol{c}, \rho)$, di centro $\boldsymbol{c}$ e di raggio ρ, viene costruito una bounding-box cubica di centro $\boldsymbol{c}$ e di lato 2ρ, grazie alla quale costruire, tramite il metodo prima delineato, un soprainsieme dei punti contenuti all'interno della sfera b; l'insieme dei punti strettamente contenuti all'interno della sfera può essere quindi individuato semplicemente eliminando da questo soprainsieme tutti i punti $\boldsymbol{p}$ la cui distanza dal centro della sfera $\boldsymbol{c}$ risulti strettamente maggiore di ρ.

3.3.2 *k*-neighborhood

Un problema ampiamente diffuso nel contesto della ricostruzione di superfici è quello di rintracciare il sottoinsieme dei punti del dataset più vicini ad un qualche altro punto specificato come parametro; la dimensione di tale sottoinsieme dipende da diversi fattori, quali per esempio il particolare problema da risolvere o il rumore presente nei dati, ed è pertanto specificato anch'esso come parametro. La formulazione più generale del problema è cioè, dati

k e $\boldsymbol{p}$, costruire l'insieme dei k punti appartenenti al dataset che risultino più vicini a $\boldsymbol{p}$; tale insieme viene anche detto il *k-neighborhood* di $\boldsymbol{p}$ e viene indicato con $k\text{-}Nbhd(\boldsymbol{p})$. Se formulassimo il problema in termini leggermente diversi, ci renderemmo conto che in realtà una sua (parziale) soluzione è già stata delineata: vale a dire determinare il raggio minimo ρ che una sfera $b = (\boldsymbol{p}, \rho)$ centrata in $\boldsymbol{p}$ deve avere per contenere esattamente k punti. Questa seconda formulazione ci suggerisce cioè di impiegare la procedura in precedenza descritta per risalire ai punti del dataset inclusi all'interno di una data sfera; questa procedura richiedeva però che fosse noto della sfera non solo il centro $\boldsymbol{p}$, ma anche il suo raggio ρ. Data l'impossibilità di definire un unico valore di ρ che garantisca di individuare sempre i k punti richiesti, la soluzione da noi implementata assegna a ρ un valore sensato ρ_0 di partenza e controlla quindi che, all'interno della sfera così definita, cadano almeno k punti. Nel caso tale condizione non fosse verificata, il raggio della sfera viene incrementato di un fattore di crescita α fintantoché all'interno della sfera risultante non cadano almeno k punti. Nell'implementazione attuale di questa procedura, sia il raggio iniziale ρ_0 sia il fattore di crescita α sono impostati alla diagonale dei voxel di dimensione minore, quelli cioè associati alle foglie dell'albero a profondità maggiore. Una soluzione probabilmente più accorta sarebbe potuta essere quella di modificare a run-time il fattore di crescita α in funzione dei numeri di flop fino a quel momento incontrato, nel senso che il valore α viene automaticamente modificato dal framework in funzione del numero di iterazioni necessarie alla procedura per trovare i k punti più vicini al query point $\boldsymbol{p}$.

3.4 Estensione *out-of-core*

Nella trattazione fino ad ora condotta si è tacitamente assunto che l'intero dataset possa essere interamente mantenuto in memoria principale: in realtà

tale assunzione può ritenersi valida solo per dataset di dimensioni ridotte, derivanti o da scansioni di oggetti non troppo grossi o da acquisizioni non troppo accurate. Questa assunzione inizia a venir meno non appena le dimensioni del dataset raggiungano, almeno in ordine di grandezza, la dimensione della memoria principale: giusto per dare l'idea delle dimensioni che il problema può raggiungere, facciamo osservare che in genere il numero di punti compresi in ogni range map varia dai 10K per gli scanner a triangolazione, capaci di individuare allo stato attuale della tecnologia ben dieci campioni per mm^2, fino a 10M per gli scanner a tempo di volo. Se a ciò si aggiunge che il numero di scansioni necessarie a generare un modello tridimensionale completo può variare anch'esso tra 10 fino a 1K per oggetti complessi o di grosse dimensioni, un semplice calcolo porta a concludere che la memoria necessaria solamente a caricare una nuvola di punti così densa deve essere superiore ai 30GB. Processare questi dataset è ancora possibile, o suddividendo il problema iniziale in diversi problemi di dimensioni minori, oppure esplicitamente gestendo a programma l'eventualità che la risorsa memoria principale esaurisca. La prima soluzione è certamente più semplice, ma ha delle grosse limitazioni: innanzitutto richiede che a priori siano note alcuni caratteristiche del dataset, cosicché possa essere suddiviso coerentemente in problemi di dimensione minore; inoltre necessita di una fase di preprocessing dei dati, durante la quale si rende probabilmente necessario l'intervento di un operatore umano; tra l'altro, non sempre risulta possibile ottenere delle partizioni sensate del problema iniziale. La seconda alternativa, sebbene richieda un maggior sforzo dal punto di vista progettuale ed implementativo, fornisce una soluzione elegante al problema senza vincolare tra l'altro la dimensione del dataset alla memoria disponibile su un dato calcolatore. L'integrazione dell'estensione out-of-core già all'interno del framework offre inoltre un ulteriore vantaggio, quello cioè di aumentare ulteriormente l'astrazione con cui dall'esterno viene percepito il dataset, vale a dire come una nuvola di punti

indicizzata interamente e costantemente mantenuta in memoria principale. Tra l'altro la scelta di delegare al framework la gestione dell'out-of-core è ulteriormente giustificata dall'osservazione che l'accesso ai punti del dataset avviene solo ed esclusivamente attraverso le spatial query e quindi le handle: poiché è noto il punto d'accesso ai dati, è immediato conoscere quali siano i dati che verranno a breve termine riferiti ed è pertanto possibile assicurare che almeno questi dati siano effettivamente presenti in memoria principale. L'idea cioè alla base dell'estensione out-of-core è quella di mantenere interamente in memoria principale almeno la struttura dati, opportunamente modificata perché tenga traccia anche dei dati realmente caricati, e di gestire attraverso questa una sorta di *cache* sui dati. Di seguito saranno introdotte le modifiche da apportare alla struttura dati ed agli algoritmi che su di essa operano affinché possa essere correttamente gestita l'estensione out-of-core. Per lealtà e correttezza precisiamo che al momento della stesura di questa tesi l'estensione out-of-core così come ora la esporremo è ancora in fase di sviluppo.

La prima modifica necessaria riguarda la fase di caricamento delle range map, durante la quale viene determinata la bounding-box globale contenente l'intero dataset. Come spiegato in precedenza, a partire da questa fase ogni range map è rappresentata da una handle, la quale, al momento della sua creazione, provvede al caricamento dei punti presenti nella range map associatole, alla loro trasformazione ed all'aggiornamento della bounding-box globale, cosicché la rispettiva range map ne risulti compresa; quando tutte le range map sono state caricate, la bounding-box globale risulta determinata e le varie handle provvedono all'ordinamento dei punti contenuti nelle rispettive range map ed alla loro indicizzazione. Per le assunzioni alla base dell'estensione out-of-core, risulta improponibile mantenere caricate in memoria principale tutte le handle durante le fasi di analisi delle range map e di indicizzazione del dataset; tra l'altro una volta che una data handle ha aggiornato

la bounding-box globale, i punti in essa contenuti di fatto non vengono più riferiti fino alla successiva fase di indicizzazione del dataset. Questa osservazione suggerisce cioè di scaricare momentaneamente dalla memoria principale i punti associati ad ogni handle subito dopo che questa abbia aggiornato la bounding-box globale; poiché però la perdita di tali punti si rivelerebbe estremamente svantaggiosa, in quanto derivanti da una fase di analisi piuttosto articolata, ogni handle provvede a salvarli in un file temporaneo di cui tiene traccia, in modo tale da poterli poi recuperare facilmente. Quando tutte le range map sono state processate e, di conseguenza, il volume occupato dall'intero dataset risulta ben definito ha inizio il processo partizionamento del volume e di indicizzazione del dataset. Quest'ultimo, per come lo si è implementato, può essere completato indicizzando separatamente l'insieme dei punti associati alle varie handle, consentendoci quindi, anche durante la fase di indicizzazione, di mantenere caricato in memoria principale solamente un insieme di punti alla volta. Prima di procedere all'indicizzazione del proprio insieme di punti, ogni handle dovrà prima recuperarlo dal file temporaneo in cui lo aveva precedentemente salvato. A questo punto, l'indicizzazione avviene esattamente come descritto in precedenza: l'insieme dei punti viene cioè ordinato secondo lo z-order loro associato e, contestualmente, vengono aggiunte alle foglie dell'octree le informazioni necessarie per l'indicizzazione del dataset. Una volta che il proprio insieme di punti è stato indicizzato, la handle provvede a scaricarlo dalla memoria principale ed a salvarlo in un nuovo file temporaneo, all'interno del quale i punti sono ordinato rispetto allo z-order loro associato, in modo tale da poter ancora sfruttare le informazioni di indicizzazione precedentemente descritte.

Quando questa fase termina, l'intero dataset risulta indicizzato ed in memoria principale risiede solamente l'octree con le informazioni di indicizzazione: il caricamento dei punti avviene quindi gradualmente in seguito alle spatial query che vengono inoltrate al framework fino al raggiungimento di

una dimensione massima di occupazione, superata la quale è necessaria la deallocazione dalla memoria di quei punti non più utili. La tecnica che abbiamo scelto per la gestione dell'estensione out-of-core è basata sul *memory mapped I/O* grazie alla quale è possibile svincolarsi dalle operazioni esplicite di input/output, che vengono mascherate dall'astrazione della memoria virtuale, e concentrarsi invece sul problema di come mantenere in memoria principale solo informazioni effettivamente utili. L'idea alla base dell'estensione out-of-core proposta è quella di realizzare una sorta di *cache* sull'insieme dei punti, attraverso la quale garantire che in memoria principale sia caricato almeno quel sottoinsieme del dataset su cui l'algoritmo di fusione sta effettivamente lavorando. Tale porzione del dataset è sicuramente nota al framework, in quanto le spatial query sono l'unico modo attraverso cui l'algoritmo di fusione può accedere ai punti del dataset. Conoscendo quindi la porzione di volume che sta venendo processata è immediato realizzare la politica di caching per mezzo dello stesso octree. A tal fine è necessario estendere le informazioni di indicizzazione presenti nei nodi dell'albero con un ulteriore flag attraverso il quale marcare l'effettiva presenza in memoria principale dei punti contenuti all'interno del voxel associato ad un dato nodo. L'unica modifica da apportare alle procedure precedentemente delineate per la gestione delle spatial query concerne la possibilità che alcuni punti non siano caricati in memoria. In questo caso, dopo aver individuato l'insieme dei nodi di profondità k il cui voxel risulti utile per la generazione della risposta, l'insieme dei punti presenti all'interno di tali voxel viene caricato in memoria principale. In seguito a questa operazione è necessario quindi aggiornare la cache, cosicché eventuali punti non più utili possano essere scaricati dalla memoria principale. Tale cache viene mantenuta attraverso una lista gestita con politica LRU (*least recently used*) definita direttamente sui nodi dell'albero: al caricamento di un insieme di punti o all'utilizzo di un insieme già caricato in memoria principale, i nodi all'interno dei cui voxel cadano questi

punti vengono spostati in cima alla lista; nel momento stesso in cui tale lista dovesse superare una certa dimensione, i punti contenuti nei voxel dei nodi in coda alla lista vengono scaricati dalla memoria principale e il flag di tali nodi aggiornato conseguentemente. La profondità k dei nodi sui quali realizzare la politica di allocazione è un parametro cruciale per il conseguimento di una struttura dati scalabile: implementare infatti la politica di allocazione alle foglie dell'octree risulterebbe in un'eccessiva frammentazione della memoria nonché ad un eccessivo numero di operazioni di lettura/scrittura su file, che potrebbero diventare un serio collo di bottiglia; del resto, implementare tale politica ai nodi eccessivamente vicini alla radice causerebbe un'allocazione di grossi blocchi di memoria, facendo quindi perdere parte dei vantaggi derivanti dall'utilizzo di questa estensione.

La possibilità di caricare facilmente in memoria solo l'insieme di punti contenuti all'interno del voxel associato ad un dato nodo è garantita dalla modalità con cui il dataset è rappresentato all'interno del framework. Ricordiamo a tal proposito che le varie range map sono rappresentate all'interno del framework con delle handle, ognuna delle quali mantiene il rispettivo insieme di punti ordinato secondo lo z-order e memorizzato su memoria secondaria; dato un nodo n dell'octree, grazie alle informazioni di indicizzazione è immediato risalire all'insieme di handle di cui almeno un punto cada all'interno del voxel associatogli e quindi, per ognuna di tali handle, all'intervallo esatto di punti che risultino contenuti all'interno di tale voxel. Pertanto, il caricamento dei punti contenuti nel voxel associato ad un dato nodo si riduce ad una sequenza di letture da file: ognuna di tali letture è mirata, nel senso che per ogni handle è noto l'intervallo esatto all'interno dell'ordinamento dove sono memorizzati i punti inclusi nel voxel. Come accennato in precedenza, tali letture non vengono effettuate esplicitamente, ma vengono mascherate con il meccanismo della memoria virtuale: grazie a questa astrazione si può operare sull'insieme di punti associati ad una handle come se fossero tutti

caricati in memoria; in realtà, di tutti i punti presenti in un dato file, solo un piccolo sottoinsieme risulterà effettivamente caricato in memoria principale.

L'uso combinato della tecnica di ordinamento z-order per i file in memoria virtuale e della politica LRU di gestione della memoria garantisce inoltre il rispetto di una proprietà estremamente importante ai fini della realizzazione di un framework efficiente, vale a dire la proprietà di località nel tempo degli accessi. Questa proprietà è un caso particolare di un principio che statisticamente viene verificato durante l'esecuzione di un qualsiasi programma: il principio della *località*, secondo cui i riferimenti generati da un processo tendono ad accentrarsi puntualmente ad un insieme ridotto di informazioni. Nel nostro caso infatti durante l'analisi di una determina porzione del volume l'insieme di dati che l'algoritmo di integrazione andrà ad utilizzare sono solamente quelli nell'immediato intorno di tale volume: grazie alla politica LRU il sottoinsieme del dataset che puntualmente risulta caricato in memoria principale tende a coincidere con quello che l'algoritmo sta effettivamente utilizzando.

Capitolo 4

Estrazione della superficie

Ogni nuvola di punti descrive implicitamente una superficie che può essere tradotta per mezzo di un algoritmo di fusione in una descrizione esplicita, quale in genere una mesh simpliciale: fra queste due entità esiste un legame molto stretto, nel senso che le proprietà esibite dalla superficie estratta dipendono fortemente dalle garanzie offerte dall'algoritmo di fusione con cui tale superficie è stata generata. Questo legame può essere impiegato a proprio vantaggio durante la fase di analisi delle soluzioni esistenti, in quanto consente di identificare quella che fra tutte maggiormente risponde alle proprie esigenze. La fase preliminare di analisi dei requisiti condotta all'inizio del nostro lavoro ha evidenziato quali proprietà avrebbe dovuto possedere l'algoritmo che avremmo poi implementato: irrinunciabilmente si sarebbe dovuto mostrare robusto rispetto agli errori derivanti dalle fasi precedenti di acquisizione e di allineamento ed efficiente, in termini sia di occupazione di memoria sia di complessità computazionale; inoltre, la superficie che avrebbe dovuto generare sarebbe dovuta essere precisa ed accurata, nel senso che non avrebbe dovuto essere affetta dai tipici problemi condivisi da molti algoritmi di fusione. La proprietà di bassa occupazione di memoria deriva dall'utilizzo del framework con estensione out-of-core descritto nel capitolo precedente; le altre proprietà, come già osservato in 2.3, caratterizzano invece tutti gli algoritmi che adottino una rappresentazione implicita della superficie. Fra i

vari algoritmi appartenenti a questa famiglia la nostra scelta è ricaduta sul metodo point-set descritto in 2.2.5, per via dell'estrema genericità e della notevole efficacia: ricordiamo infatti che tale metodo consente la definizione di un'intera famiglia di superfici, le extremal surface, semplicemente variando formulazione del vector field e dell'energy function. In tutti gli approcci basati su rappresentazioni point-set tali funzioni vengono utilizzate all'interno di una procedura di proiezione di punti sulla superficie: l'insieme dei punti che "convergono" sulla superficie costituiscono infine l'extremal-surface. È interessante far notare che le più importanti formulazioni di tale procedura mancano di una formale dimostrazione di correttezza, nel senso che seppur in pratica gli errori da loro introdotti sono trascurabili, nessuna di esse genera punti che giacciono esattamente sulla superficie. Poiché una garanzia di correttezza ci è sembrata un ottimo punto di partenza su cui costruire il nostro algoritmo, la procedura da noi adottata è quella delineata in [5], l'unica di cui ne sia stata rigorosamente provata la convergenza.

4.1 Superfici Point-Set per *surfel*

I metodi point-set individuano la superficie implicata da una nuvola di punti per mezzo dell'interazione di due diverse funzioni, il vector field e la energy function: formulazioni diverse di queste funzioni consentono di generare superfici con proprietà diverse ma appartenenti sempre ad una stessa famiglia di superfici implicite, quella delle extremal surface. Delle definizioni proposte in letteratura per queste due funzioni, molte sono formulate esclusivamente sull'insieme dei punti presenti nel dataset. Tuttavia, come già detto nei capitoli precedenti, l'input di un algoritmo di fusione è qualcosa di più di una semplice nuvola di punti, nel senso che in genere ad ogni punto sono associate ulteriori informazioni o almeno una normale: queste ultime sono del resto sicuramente disponibili se il dataset è stato o ricavato dall'acquisizione

di un oggetto reale oppure generato per conversione in nuvola di punti di un modello tridimensionale già esistente.

Una extremal surface concepita per trarre beneficio dalla definizione delle normali ai punti è la superfice point-set per *surfel* (PSS)[5], dove per surfel si intende la generica coppia punto-normale $(\boldsymbol{x}_i, \vec{\boldsymbol{n}}_i)$ del dataset. Tale superficie, in quanto facente parte della famiglia delle extremal surface, viene sempre individuata per mezzo dell'interazione delle due funzioni energy function e e vector field n, la cui definizione viene questa volta data in termini non solo dei punti ma anche delle normali loro associate. Come illustreremo tra breve tale definizione è estremamente intuitiva e conferisce tra l'altro alla superficie ricostruita una maggior robustezza, rispetto ad un'altra extremal surface generata utilizzando i soli punti, di fronte a problemi di sottocampionamento o di irregolarità nella distribuzione dei dati.

La definizione del vector field n segue l'intuizione che la sua valutazione in un punto $\boldsymbol{p}$ nello spazio debba in qualche modo riflettere l'andamento della normale alla porzione di superficie più vicina a tal punto: il campo vettoriale può cioè essere calcolato a partire dalle normali presenti nel dataset, imponendo che la direzione associata ad un punto $\boldsymbol{p}$ sia influenzata maggiormente dai punti nel dataset a lui più vicini. Questa condizione viene imposta attraverso la seguente weighting function:

$$\vartheta_N(\boldsymbol{p}, \boldsymbol{x}_i) = \frac{e^{-\frac{\|\boldsymbol{p}-\boldsymbol{x}_i\|^2}{h^2}}}{\sum_j e^{-\frac{\|\boldsymbol{p}-\boldsymbol{x}_j\|^2}{h^2}}},$$

una gaussiana normalizzata formulata in termini della distanza, della cui importanza si parlerà nel seguito. Il vector field viene quindi definito come media pesata delle normali ai surfel, vale a dire:

$$n_{\mathrm{PSS}}(\boldsymbol{p}) = \sum_i \vec{\boldsymbol{n}}_i \vartheta_N(\boldsymbol{p}, \boldsymbol{x}_i).$$

Anche la energy function e viene formulata in maniera molto intuitiva

come una sorta di distanza non-segnata dalla superficie espressa in termini delle posizioni e delle normali dei surfel. Poiché sarebbe preferibile assegnare una distanza maggiore ai punti disposti lungo le normali dei surfel, la formulazione dell'energy function si appoggia alla distanza di Mahalanobis, una sorta di distanza euclidea avente però supporto ellittico anziché sferico:

$$\mathrm{dist}_M(\boldsymbol{p}, \boldsymbol{x}_i, \vec{\boldsymbol{n}}_i) = \big((\boldsymbol{p} - \boldsymbol{x}_i) \cdot \vec{\boldsymbol{n}}_i\big)^2 + c \left\| (\boldsymbol{p} - \boldsymbol{x}_i) - \big((\boldsymbol{p} - \boldsymbol{x}_i) \cdot \vec{\boldsymbol{n}}_i\big)\vec{\boldsymbol{n}}_i \right\|^2,$$

dove c è uno fattore di scala che incide sulla forma dell'ellisse; più precisamente con $c = 1$ la distanza di Mahalanobis è equivalente alla distanza euclidea fra il punto $\boldsymbol{p}$ ed il campione $\boldsymbol{x}_i$, mentre con $c = 0$ corrisponde alla distanza di $\boldsymbol{p}$ dal piano passante per $\boldsymbol{x}_i$ orientato in $\vec{\boldsymbol{n}}_i$. La risultante energy function è:

$$e_{\mathrm{PSS}}(\boldsymbol{p}, \vec{\boldsymbol{n}}) = e_{\mathrm{PSS}}(\boldsymbol{p}) = \sum_i \mathrm{dist}_M(\boldsymbol{p}, \boldsymbol{x}_i, \vec{\boldsymbol{n}}_i)\ \vartheta_N(\boldsymbol{p}, \boldsymbol{x}_i).$$

La superficie point-set per surfel è infine determinata, similmente a tutte le altre extremal surface, individuando l'insieme dei punti di minimo della funzione energia lungo le direzioni imposte dal campo vettoriale, vale a dire:

$$S = \Big\{ \boldsymbol{p} \,|\, \boldsymbol{p} \in \operatorname*{arglocalmin}_{\boldsymbol{q} \in \ell_{\boldsymbol{p}, n_{\mathrm{PSS}}(\boldsymbol{p})}} e_{\mathrm{PSS}}\big(\boldsymbol{q}, n_{\mathrm{PSS}}(\boldsymbol{p})\big) \Big\}.$$

La procedura che permette di identificare i punti della superfice PSS implicata da n_{PSS} ed e_{PSS} è formulata in modo ricorsivo ed è illustrata in figura 4.1. Ad ogni iterazione si calcola la normale $\vec{\boldsymbol{n}}_i = n_{\mathrm{PSS}}(\boldsymbol{p}_i)$ definita dal vector field in $\boldsymbol{p}_i$ e si considera la linea $\ell_{\boldsymbol{p}_i, \vec{\boldsymbol{n}}_i}$ individuata dalla sua direzione: il minimo locale sull'insieme di punti $\boldsymbol{q} \in \ell_{\boldsymbol{p}, n(\vec{\boldsymbol{p}})}$ dell'energy function $e_{\mathrm{PSS}}(\boldsymbol{q}, n(\boldsymbol{p}))$ identifica il punto $\boldsymbol{p}_{i+1}$ da utilizzare all'iterazione successiva. Poiché ad ogni iterazione il campo vettoriale viene ricalcolato come $n(\boldsymbol{p}_{i+1})$, l'energy function e_{PSS} ad ogni passo risulterà minore, cosicché la procedura raggiungerà probabilmente la convergenza. Il minimo dell'energia lungo la linea $\ell_{\boldsymbol{p}, n_{\mathrm{PSS}}(\boldsymbol{p})}$

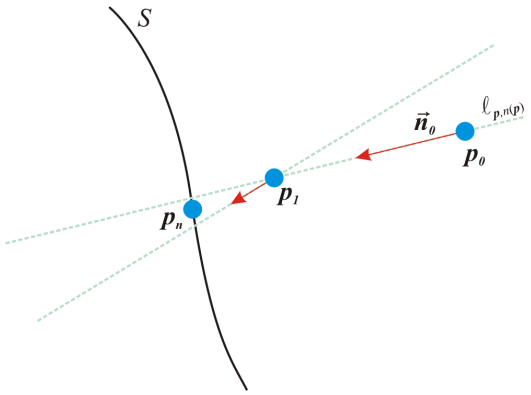


Figura 4.1: *Procedura per la definizione della superficie PSS. Il punto* $\boldsymbol{p}_0$ *viene proiettato su un minimo locale lungo la linea* $\ell_{\boldsymbol{p}_0,\vec{\boldsymbol{n}}_0}$*, rappresentato dal punto di intersezione delle due linee tratteggiate; tale punto diventa* $\boldsymbol{p}_1$*. Quando il processo converge, il punto* $\boldsymbol{p}_n$ *giace sulla superficie.*

passante per $\boldsymbol{p}$ in direzione $n_{\mathrm{PSS}}(\boldsymbol{p})$ può essere individuato grazie ad un'opportuna riformulazione del problema attraverso il metodo di Brent [28] di minimizzazione non-lineare in una sola dimensione: il punto da proiettare sulla superficie viene espresso come $\boldsymbol{q} = \boldsymbol{p} + \tau n_{\mathrm{PSS}(\boldsymbol{p})}$, e l'energy function $e(\boldsymbol{q})$ diventa quindi una funzione nella sola incognita τ.

La possibilità di identificare i punti della superficie come i punti di minimo dell'energia lungo le direzioni del campo vettoriale è una conseguenza della particolare weighting function adottata. Tale funzione doveva necessariamente essere formulata in termini delle distanze fra il punto di valutazione $\boldsymbol{p}$ e gli elementi del dataset al fine di ottenere un vector field ed una energy function che fossero maggiormente influenzate dai surfel più vicini a $\boldsymbol{p}$. Se si fosse adottata come weighting function una semplice gaussiana ϑ, i pesi generati sarebbero stati sempre minori con l'allontanarsi dalla nuvola di punti e la risultante energy function avrebbe avuto lontano dalla superficie dei valori molto prossimi allo zero, e questo indipendentemente da come il

vector field n fosse stato definito: per definizione di extremal surface, molti di questi minimi locali sarebbero potuti essere scambiati per componenti della superfice PSS generata. Come suggerito in [2], questo ambiguità può essere elusa semplicemente sostituendo la gaussiana ϑ con la sua versione normalizzata ϑ_N, la quale permette di generare un'energy function e che, sempre indipendentemente dai valori del campo vettoriale, assume valori crescenti all'aumentare della distanza della superficie. Anche nel caso degenere in cui il campo vettoriale $n(\boldsymbol{p})$ risultasse all'incirca parallelo alla normale del surfel più vicino a $\boldsymbol{p}$, l'energy function e_{PSS} risulta essere una stima della distanza segnata di $\boldsymbol{p}$ dalla nuvola di punti: in questo caso il suo minimo lungo la linea $\ell_{\boldsymbol{p},n_{\text{PSS}}(\boldsymbol{p})}$ risulta essere prossimo alla nuvola di punti, in quanto proprio in questa zona l'energia raggiunge il suo minimo.

Talvolta ad ogni surfel del dataset è associato anche un peso che può essere visto come una misura della qualità del surfel e può essere usato di fronte a problemi di irregolarià del campionamento o di scarsa attendibilità di alcune acquisizioni. Se h_i è il peso associato al surfel $(\boldsymbol{x}_i, \vec{\boldsymbol{n}}_i)$, allora la seguente formulazione della funzione dei pesi:

$$\vartheta_N(\boldsymbol{p}, \boldsymbol{x}_i) = \frac{e^{-\frac{\|\boldsymbol{p}-\boldsymbol{x}_i\|^2}{h_i^2}}}{\sum_j e^{-\frac{\|\boldsymbol{p}-\boldsymbol{x}_j\|^2}{h_j^2}}}$$

permette di generare un vector field ed una energy function che risultino influenzate anche dalla qualità dei singoli surfel.

4.2 Dai punti alla superficie

I metodi di integrazione point-set descrivono la superficie del modello per mezzo di una procedura capace di proiettare un qualsiasi punto nello spazio sulla superficie implicata dal dataset in input: l'insieme dei punti generati per mezzo di tale procedura formano una nuova nuvola di punti che, a

differenza di quella in input, è garantito giaccia esattamente sulla superficie dell'oggetto. Sebbene questa garanzia si riveli di fondamentale importanza nella definizione di un algoritmo di fusione, tale procedura non è sufficiente da sola per generare una descrizione esplicita della superficie, in quanto nessuna informazione topologica viene da essa estrinsecata alla fine del processo di proiezione dei punti. La nostra proposta consiste nel combinare le garanzie offerte dalle procedure di proiezione dei punti integrate nei metodi point-set con la semplicità e la ricchezza descrittiva offerta dalle mesh simpliciali.

Il modello tridimensionale generato dal nostro algoritmo è descritto esplicitamente attraverso la mesh di triangoli che ne approssima la superficie: l'insieme di questi triangoli viene generato da un algoritmo di triangolarizzazione durante la fase di analisi del volume occupato dal dataset. In particolare, la generazione della triangolarizzazione della superficie può essere effettuata nella nostra implementazione da due diversi algoritmi, entrambi varianti del noto *Marching Cubes* proposto da Lorensen e Cline [22]: di questi algoritmi, il primo [21] garantisce la generazione di una topologia consistente grazie all'utilizzo di una *lookup table* estesa per la disambiguazione dei casi, mentre il secondo, l'*Extended Marching Cubes* [19], consente di preservare nella superficie ricostruita le feature presenti nel modello iniziale limitando inoltre la generazione di artefatti in prossimità di spigoli ed angoli. Poiché varianti del marching cubes, entrambi gli algoritmi condividono le attraenti proprietà di semplicità e robustezza tipiche di quest'algoritmo. La generazione della triangolarizzazione avviene infatti progressivamente, campionando ad intervalli discreti, lungo gli spigoli di una griglia, un qualche campo scalare a valori reali, e quindi generando per ogni cella della griglia l'insieme di triangoli che meglio approssima la porzione della superficie che la attraversa: l'insieme di tutti i triangoli così generati costituisce infine la superficie dell'oggetto ricostruito. Sebbene questo approccio risulti concettualmente semplice, tuttavia nel nostro caso la sua implementazione è risultata tutt'al-

tro che immediata, a causa sia dell'assenza di un campo scalare segnato sia delle dimensioni del dataset e della conseguente impossibilità di mantenere interamente in memoria tale campionamento.

Il problema dell'assenza del campo scalare è stato risolto notando che in realtà dalla procedura di proiezione dei punti sulla superficie può essere ricavata una funzione distanza segnata. Per quanto detto in precedenza, questa procedura consente di proiettare un punto $\boldsymbol{p}$ nello spazio sulla porzione di superficie a lui più vicina lungo la direzione definita dal vector field n in $\boldsymbol{p}$. L'applicazione cioè di questa procedura al punto $\boldsymbol{p}$ restituisce una coppia punto-direzione $(\boldsymbol{p}^\star, \vec{\boldsymbol{n}}^\star)$ che rappresentano rispettivamente la proiezione del punto $\boldsymbol{p}$ sulla superficie e la direzione lungo cui $\boldsymbol{p}$ è stato proiettato: più precisamente, poiché la procedura di proiezione è iterativa fino alla individuazione di un punto stazionario, la direzione $\vec{\boldsymbol{n}}^\star$ restituita è quella calcolata nel ciclo in cui viene raggiunta la convergenza. La coppia di valori $(\boldsymbol{p}^\star, \vec{\boldsymbol{n}}^\star)$ permette di individuare il piano P passante per $\boldsymbol{p}^\star$ ed orientato in direzione $\vec{\boldsymbol{n}}^\star$ che, poiché generato sul risultato della procedura di proiezione applicata a $\boldsymbol{p}$, risulta essere la miglior approssimazione locale della porzione di superficie più prossima a $\boldsymbol{p}$. Tale approssimazione può essere quindi impiegata per individuare una stima della distanza di un punto $\boldsymbol{p}$ dalla superficie: questa viene approssimata con la distanza fra $\boldsymbol{p}$ stesso e la sua proiezione sul piano P individuato per mezzo dei risultati della procedura di proiezione a lui applicata. Per come definito il vector field in 4.1, e cioè come media pesata delle normali dei surfel più vicini al punto $\boldsymbol{p}^\star$, la direzione $\vec{\boldsymbol{n}}^\star$ individuata dalla procedura di proiezione non manca di verso e dunque la distanza così calcolata per un punto $\boldsymbol{p}$ dalla superficie risulta necessariamente segnata.

La funzione distanza segnata appena definita è stata da noi applicata per generare il campo scalare a valori reali necessario agli algoritmi di triangolarizzazione per l'individuazione della superficie. Tali algoritmi suddividono il volume in celle e campionano il valore della funzione distanza segnata ai

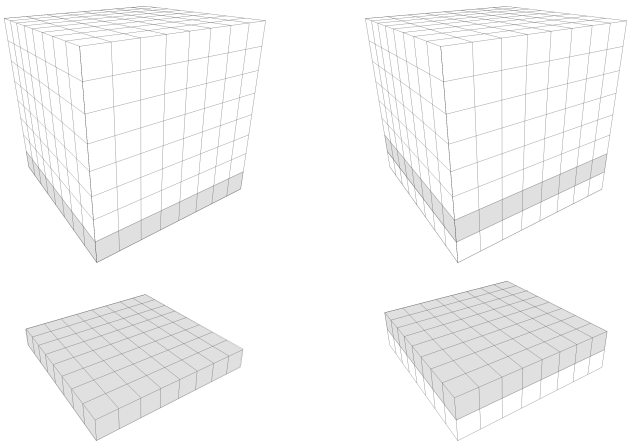


Figura 4.2: *Il volume viene processato per fette perpendicolari all'asse y e per valore crescente di ordinata: ad ogni iterazione vengono recuperate parte delle informazioni calcolate durante l'analisi della fetta visitata al passo precedente.*

corner di queste ultime: le celle ai cui corner si verifichi almeno un cambiamento di segno della funzione distanza segnata risultano attraversate dalla superficie; un'opportuna triangolarizzazione basata sui valori ai corner delle celle identifica quindi l'isosuperficie di valore zero. All'interno del nostro algoritmo il campionamento della funzione distanza segnata è effettuato ai corner dei voxel associati ai nodi dell'octree di una prefissata altezza. Questa scelta ha permesso di generare modelli tridimensionali di qualità differente a seconda della profondità di visita dell'octree: modelli più dettagliati vengono generati dal campionamento della funzione distanza segnata ai corner dei voxel associati alle foglie, mentre modelli meno particolareggiati possono essere generati estraendo la superficie che attraversa i voxel associati ai nodi di altezza inferiore. Tuttavia il campionamento della funzione distanza segnata sulla griglia indotta dall'octree per una prefissata altezza può risultare proibitivo, soprattutto quanto il dataset risulti composto da parecchie range map:

più che altro, al fine della ricostruzione del modello tridimensionale, è inutile effetture il campionamento del dataset ai corner di *tutti* i voxel e soprattutto risulta inutile mantenere interamente in memoria tale campionamento. La nostra implementazione, per via dei vincoli di efficienza e di bassa occupazione di memoria principale, effettua l'analisi del volume per fette e, di ognuna di queste, esamina solamente quei voxel che probabilmente conterranno una porzione di superficie. Più precisamente, una volta nota la profondità massima h di visita dell'albero, è nota anche la massima risoluzione 2^{3h} della griglia con cui l'octree partiziona il volume: la visita di quest'ultimo avviene per fette perpendicolari all'asse y per valori crescenti di ordinata (figura 4.2). In questo modo in memoria principale risultano puntualmente caricate solamente due fette di dimensione al più 2^h, necessarie a memorizzare i valori del campo scalare ai corner dei voxel appartenenti alla fetta in esame. Infatti il campionamento della funzione distanza segnata avviene solamente ai corner di un piccolo sottoinsieme dei voxel appartenenti alla fetta corrente: questi voxel vengono identificati attraverso spatial query all'octree mirate in questo caso ad individuare solamente quei voxel appartenenti alla fetta corrente al cui interno sia contenuto almeno un punto del dataset. I voxel individuati da queste query conterranno con alta probabilità una porzione di superficie: per ognuno di essi, si calcola prima la distanza segnata ai suoi corner e si procede quindi con l'eventuale triangolarizzazione dell'isosuperficie in esso contenuto. Poiché voxel adiacenti condividono quattro corner, è opportuno associare ad ognuno di essi il rispettivo valore della distanza segnata, così da ridurre quanto più possibile il tempo necessario al campionamento del campo scalare; similmente è anche necessario tener traccia dei vertici dei triangoli generati dagli algoritmi di triangolarizzazione durante l'analisi di un voxel, in quanto tali vertici saranno condivisi nella mesh ricostruita con i triangoli che approssimano la superficie passante per i voxel adiacenti. L'associazione del campo scalare ad ogni corner di un voxel così come l'associazione di un

vertice della mesh ad un suo dato edge vengono mantenute con delle tabelle hash, all'interno delle quali il corner o l'edge di ogni voxel sono identificati attraverso un intero non-segnato generato a partire dalle octree coordinate del voxel cui appartengono. L'identificativo unico associato ad ogni voxel viene ricavato utilizzando solamente le componenti x e z della suo centro espresso in octree coordinate: l'utilizzo infatti anche della componente y non aggiungerebbe nessuna informazione utile all'identificazione del voxel, dal momento che tutti i voxel che si analizzano ad ogni iterazione hanno, per via dell'ordine di visita imposto al volume, centro di uguale ordinata. Tale intero risulta dalla linearizzazione del piano cui appartengono i centri dei voxel secondo l'ordinamento imposto dalla curva space-filling di Lebesgue per uno spazio bidimensionale e può essere quindi calcolato per mezzo di semplici operazioni sui bit una volta noto l'identificativo del nodo cui un voxel è associato; a partire dal nodo `n`, il suo identificativo può cioè essere ricavato come:

```
1 int shift = max_depth-h;
2 int i     = Center(n).X>>shift;
3 int k     = Center(n).Z>>shift;
4 Id(n)     = i+(k<<h);
```

in cui `h` rappresenta la profondità di visita dell'octree scelta per la generazione del modello tridimensionale (deve ovviamente valere `h<=max_depth`).

L'implementazione da noi proposta mantiene due tabelle hash per memorizzare il campo scalare, una per i corner inferiori dei voxel e l'altra per quelli superiori, e cinque tabelle hash per memorizzare i vertici della mesh lungo gli edge dei voxel, una per gli edge allineati con l'asse y e due per quelli allineati sia con l'asse x che con l'asse z, in modo tale da separare, come per il campo scalare, gli edge inferiori da quelli superiori ai voxel. Il motivo per cui si è preferito mantenere separate le informazioni associate alle facce inferiori da quelle invece associate alle facce superiori è legato alla politica di visita del

volume: infatti le informazioni che vengono associate durante l'analisi di una fetta alle facce superiori dei voxel, potranno essere riutilizzate durante l'analisi della fetta immediatamente successiva come informazioni da associare alle facce inferiori dei voxel contenuti in questa nuova fetta. Questa soluzione risulta vantaggiosa per quanto concerne il calcolo del campo scalare, in quanto evita di ricalcolarlo laddove già noto, mentre è indispensabile per quanto riguarda i vertici della mesh, poiché garantisce che la superficie ricostruita utilizzi intercette già calcolate, evitando quindi la presenza di vertici duplicati. La scelta delle tabelle hash è dovuto all'osservazione che, ad ogni fetta, il numero di voxel attraversati dalla superficie è sempre una minima percentuale dell'insieme di tutti i voxel definibili sulla fetta. In una prima implementazione si era optato per l'utilizzo di un'array bidimensionale di dimensione pari alla risoluzione massima della griglia: nonostante la proprietà di accesso agli elementi in tempo costante, si è potuto osservare un crollo delle prestazioni, se paragonato a quelle raggiunte con l'utilizzo delle tabelle hash, già per visite dell'albero di profondità sette o otto, causato prevalentemente dalla loro inizializzazione fra l'analisi di una fetta e di quella successiva.

4.3 Hole Filling

Per come delineato fin'ora, l'algoritmo di estrazione della superficie ha una grossa limitazione, quella cioè di limitare la ricerca della superficie ai soli voxel che contengano almeno un punto del dataset. Questa soluzione è indispensabile al fine di evitare il campionamento esteso su tutta la fetta del campo scalare, calcolo decisamente oneroso se si tiene conto che viene risolto per mezzo di un algoritmo di minimizzazione non lineare; tuttavia, la mesh ricostruita in questo modo può presentare una superficie non continua, in quanto può ancora verificarsi che essa debba attraversare dei voxel all'interno dei quali non cadesse neppure un punto del dataset (figura 4.3). La soluzione

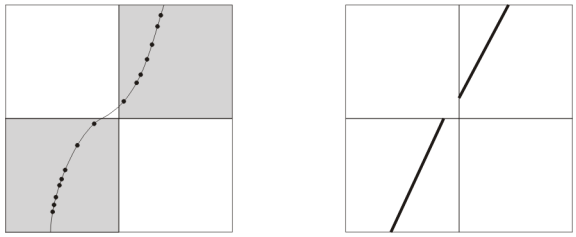


Figura 4.3: *A causa della distribuzione dei campioni rispetto alle celle della griglia, se l'algoritmo visitasse solamente quelle celle al cui interno ricade almeno un punto del dataset (celle in grigio), la superficie generata risulterebbe composta da due componenti sconnesse.*

da noi fornita a questo problema si basa sulla possibilità di conoscere per ogni voxel se è stato già visitato, se dovrà essere visitato oppure se non appartiene all'insieme dei voxel che verranno visitati. L'analisi di ogni fetta viene cioè completata in diversi *passi di espansione*, vale a dire un numero massimo di iterazioni specificato dall'utente: durante la prima di tali iterazioni, vengono visitati solamente i voxel restituiti dal framework in risposta alle spatial query rivoltegli, mentre ai passi successivi viene effettuata l'analisi dei soli voxel marcati all'iterazione immediatamente precedente. I voxel che devono essere visitati, così come quelli che sono già stati visitati sono mantenuti in due insiemi differenti: se durante la triangolarizzazione della superficie passante per un dato voxel viene rilevata un'intercetta lungo un edge condiviso con dei voxel non appartenenti né all'insieme delle celle visitate né all'insieme delle celle da visitare al passo corrente, allora tutti i voxel che contengono tale edge sono marcati ed aggiunti alla lista di voxel da analizzare al passo successivo. Facendo riferimento alla figura 4.4, durante il primo passo l'algoritmo visita solamente le celle contenenti almeno un punto del dataset (a sinistra, celle in grigio); quando durante questa iterazione vengono calcolate le intercette in rosso, poiché queste ultime sono disposte su un edge condiviso con una

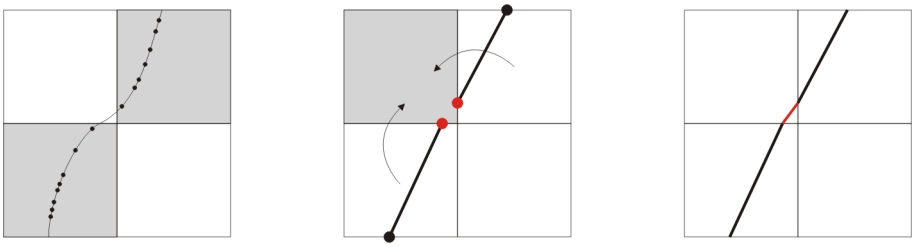


Figura 4.4: *Al fine di generare una unica superficie continua, l'analisi di ogni fetta del volume è completata in un numero massimo w di iterazioni: durante la prima iterazione vengono analizzate solamente quelle celle al cui interno ricada almeno un punto del dataset (a sinistra, celle in grigio), mentre nelle iterazioni successive le celle da processare sono quelle marcate durante l'iterazione immediatamente precedente (cella in grigio al centro).*

cella ancora nono visitata, l'algoritmo può marcare quest'ultima (al centro, cella in grigio), al fine di garantire la ricostruzione di una superficie continua (figura a destra).

Tale approccio ha permesso la generazione di superfici chiuse già per dataset di piccole dimensioni e con un numero di passi di espansione limitato (in genere uno o due). L'utilizzo però di tale modifica senza nessuna precauzione può portare alla generazione di artefatti non presenti nella nuvola di punti originale, così come è emerso anche dalla sua applicazione a dataset costruiti ad-hoc per testarne la stabilità e l'efficacia: tale tecnica tende infatti, nell'arco della visita dell'intero volume, a propagare l'analisi anche a quelle celle eccessivamente lontane dalla nuvola di punti. Il rimedio da noi proposto è stato quello di vincolare la propagazione alle sole celle che risultino distanti dalla nuvola di punti al massimo di una distanza k volte la diagonale della cella, dove k è un altro parametro passato dall'utente. Questa semplice modifica non ha in alcun modo precluso la capacità dell'algoritmo

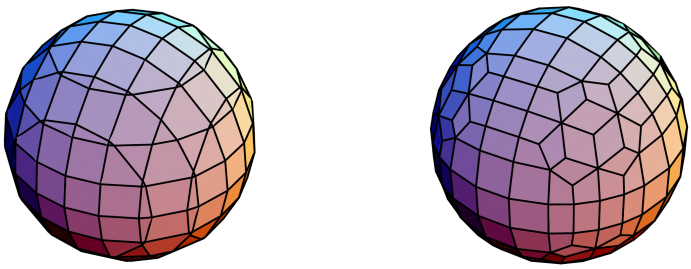


Figura 4.5: *Poligonalizzazione di una sfera generata dal marching cubes (a sinistra) e da un metodo duale (a destra).*

di generare superficie chiuse, ma anzi ha vincolato la superficie ad estendersi solo su quelle regioni probabilmente non campionabili a causa delle occlusioni presenti nell'oggetto reale.

4.4 Tentativi falliti

4.4.1 La superficie duale

I metodi *cube-based* come il Marching Cubes e le sue varianti generano una triangolarizzazione, o più in generale una poligonalizzazione, per ogni cella della griglia che risulti attraversata dalla superficie: tale poligonalizzazione viene individuata dai cambiamenti di segno della funzione distanza segnata ai corner della cella, ed i suoi vertici sono calcolati come intersezione fra la superficie e gli edge della stessa cella. Di conseguenza questi metodi generano una poligonalizzazione i cui vertici sono vincolati a stare lungo gli edge della griglia ed i cui poligoni giacciono esattamente all'interno delle celle.

I metodi *duali* [18, 17, 31] superano i vincoli cui gli algoritmi cube-based sono soggetti nel posizionamento dei vertici, e generano una superficie i cui vertici giacciono internamente alle celle ed i cui poligoni risultano essere duali agli edge della griglia (figura 4.5). Più precisamente, per ogni cella contenente

almeno un cambiamento di segno, viene generato un vertice della superficie, calcolato a partire dai vertici che un algoritmo cube-based avrebbe generato agli edge della cella, e posizionato nel punto $\boldsymbol{p}$ che minimizza la seguente funzione quadratica:

$$E(\boldsymbol{p}) = \sum_i \left(\vec{\boldsymbol{n}}_i \cdot (\boldsymbol{p} - \boldsymbol{q}_i)\right)^2$$

dove $(\boldsymbol{q}_i, \vec{\boldsymbol{n}}_i)$ sono i dati hermitiani associati agli edge della griglia; quindi, per ogni edge della griglia che esibisce un cambiamento di segno, viene generato un poligono che connette i vertici all'interno delle quattro celle che condividono questo edge. La mesh generata da questi metodi risulta quindi per costruzione duale, nel senso topologico del termine, rispetto a quella generata da un algoritmo cube-based: infatti i vertici della mesh duale corrispondono alle facce della mesh generata da un metodo cube-based, ed i vertici generati da quest'ultimi corrispondono alle facce individuate dai metodi duali. La possibilità di posizionare i vertici all'interno delle celle, anziché esclusivamente lungo gli edge della griglia, consente la generazione di poligoni con buon *aspect ratio*: di conseguenza, le mesh generate dagli algoritmi duali risultano in genere di qualità superiore rispetto a quelle generate dagli algoritmi cube-based.

Per via delle interessanti garanzie che i metodi duali offrono sulla qualità della mesh ricostruita, si è preferito non precluderci a priori la scelta sugli algoritmi di poligonalizzazione da combinare al nostro metodo di fusione, ed è stata investigata anche la possibilità di ricostruire una superficie duale. Le modifiche che si sono rese necessarie all'implementazione descritta in 4.2 per conformarla al nuovo algoritmo di poligonalizzazione sono state tutto sommato limitate, ed hanno riguardato prevalentemente la modalità attraverso cui venivano individuati i vertici della superficie ricostruita. Per evitare anche in questo caso di analizzare tutte le celle definibili all'interno del volume, si è preferito limitare l'analisi per la ricostruzione della superficie

solamente a quei voxel all'interno dei quali fosse contenuto almeno un punto del dataset. In entrambe le implementazioni realizzate nell'ambito di questo tentativo si è continuato ad utilizzare il metodo di proiezione dei punti, che però è stato impiegato non per definire la funzione distanza segnata ma per generare direttamente i vertici della superficie. In pratica la nostra idea è stata fin dal principio quella di cercare di combinare in qualche modo la proprietà della procedura di proiezione di generare punti che effettivamente giacciono sulla superficie, con l'osservazione quest'ultima probabilmente attraversa quei voxel contenenti almeno un punto del dataset. La due diverse implementazioni fornite differivano nella scelta dei punti da proiettare sulla superficie: in un primo tentativo si era provato e proiettare il centro dei voxel, previa trasformazione da octree coordinate in world coordinate, contenenti almeno un punto, mentre in una seconda soluzione il punto all'interno del voxel da proiettare era calcolato come media dei punti del dataset contenuti al suo interno pesata rispetto alla qualità loro associata. La superficie estratta dalla nuvola di punti mancava in entrambi i casi di continuità in genere a causa della presenza di crack, un problema riconosciuto anche dagli stessi ricercatori che proposero i metodi duali. Tale problema si è manifestato, indipendentemente dal numero di range map componenti il dataset, ogni volta che la superficie risultasse molto prossima e pressoché parallela ad un faccia di qualche cella; inoltre, a causa di errori di acquisizione e/o di allineamento, abbiamo potuto osservare un aumento di queste imperfezioni in tutte quelle porzioni di superficie descritte da più range map. A nostro avviso questi problemi sono generati dalla distribuzione su celle adiacenti di punti che in realtà descrivono una stessa superficie: in questo caso la superficie coincide in buona parte con la faccia in comune delle due celle, ma i punti generati durante la fase di proiezione possono cadere, a causa di errori di approssimazioni nei calcoli, in una delle due celle con pari probabilità: in questo caso trovare una triangolarizzazione dei punti della mesh così generati è tutt'al-

tro che banale, in quanto richiede l'analisi e la trattazione esaustiva di una casistica veramente ampia.

4.4.2 Vector field vincolato alla griglia

Il processo che nel nostro algoritmo porta alla creazione del modello tridimensionale segue almeno in linea di principio l'approccio adottato dagli algoritmi zero-set descritti in 2.2.1 di campionare la funzione distanza segnata ai corner di una griglia tridimensionale e quindi di generare una triangolarizzazione attraverso la quale connettere i vertici sugli edge che esibiscono un cambiamento di segno. Poiché entrambi gli algoritmi di triangolarizzazione che abbiamo impiegato sono varianti del Marching Cubes, i vertici del modello tridimensionale sono calcolati come le intersezioni fra gli edge della griglia e la superficie implicata dalla nuvola di punti: più precisamente, il calcolo della posizione del vertice lungo un dato edge avviene per interpolazione lineare dei valori ai suoi estremi della funzione distanza segnata.

Un approccio completamente diverso che abbiamo provato a seguire e che riteniamo interessante esporre è stato quello di evitare completamente il campionamento della funzione distanza segnata ai corner della griglia tridimensionale e di individuare invece le intercette lungo gli edge di quest'ultima sfruttando solo ed esclusivamente la rappresentazione implicita adottata, vale a dire l'interazione del vector field con la energy function. L'idea alla base di questo tentativo era cioè quello di applicare il metodo di minimizzazione dell'energia non rispetto al campo vettoriale così come definito in 4.1, ma rispetto ad un campo vettoriale da noi impostato e che risultava vincolato alla griglia: secondo le nostre congetture, questo tentativo avrebbe cioè dovuto portare all'identificazione delle intercette lungo un dato edge. Anche in questo caso il processo di identificazione delle intercette non veniva effettuato su tutti gli edge di tutte le possibili celle appartenenti al volume, ma piuttosto

solamente su un determinato sottoinsieme di quei voxel contenenti almeno un punto del dataset. Di tali voxel, gli edge sui quali venivano ricercate le intercette erano solamente quelli che non risultassero "eccessivamente" paralleli alla porzione di superficie implicata dalla nuvola di punti al loro interno. Poiché tale superficie ancora non esisteva, il test in base al quale sceglievamo gli edge per la ricerca delle intercette era formulato in termini della direzione media delle normali associate ai punti all'interno del voxel. Più precisamente, per ogni voxel v la normale $\vec{\boldsymbol{n}}_v$ alla porzione di superficie passante per v era approssimata dalla media pesata delle normali ai punti all'interno del voxel, vale a dire:

$$\vec{\boldsymbol{n}}_v = \frac{\sum_i \vec{\boldsymbol{n}}_i}{\| \sum_i \vec{\boldsymbol{n}}_i \|}.$$

Tale vettore veniva quindi impiegato per evitare la ricerca delle intercette lungo quegli edge che non potevano intersecare la superficie per via della direzione su cui erano allineati. Questi edge erano identificati attraverso il prodotto scalare fra la normale $\vec{\boldsymbol{n}}_v$ ed il vettore normalizzato costruito per differenza degli estremi dell'edge: se il loro prodotto scalare risultava inferiore ad un valore ζ specificato in input, il relativo edge veniva scartato in base all'intuizione che risultava troppo parallelo alla superficie per poterla intersecare. Sugli edge che invece superavano questo test venivano ricercate le intercette che sarebbero state poi utilizzate come vertici nella successiva fase di triangolarizzazione della superficie.

Per le forti analogie tra questo approccio ed il metodo effettivamente implementato, le modiche necessarie alla sua implementazione sono state veramente minime ed hanno riguardato solamente la fase di individuazione delle intercette ed il processo di minimizzazione dell'energia: quest'ultimo infatti doveva necessariamente essere ritoccato, al fine di evitare di considerare come minimi assoluti dell'energy function quei punti che in realtà erano solamente dei minimi locali lungo un dato edge. Il problema maggiore al riguardo è

stata l'individuazione di condizioni di convergenza che risultassero valide in generale: condizioni di convergenza troppo stringenti individuavano troppe poche intercette per poterci effettuare una triangolarizzazione ragionevole della superficie; d'altro canto, l'utilizzo di condizioni troppo tolleranti portava ad considerare come intercette valide anche i minimi locali. La difficoltà nell'individuare delle condizioni che risultassero valide in assoluto è legata alle interazioni tra l'energy function ed il vector field e, soprattutto, a come le varie isosuperfici si dispongono rispetto alla griglia. Capita spesso infatti che gli isocontorni della superficie risultino eccessivamente paralleli agli assi su cui la griglia è costruita: in questo caso ridurre l'analisi dell'energia ad un edge porta sicuramente all'individuazione di un minimo locale.

Capitolo 5

Risultati

I metodi e le tecniche illustrati nei capitoli precedenti sono stati utilizzati per la realizzazione di *Octree Merger* (nel seguito OM), un tool destinato all'integrazione di range map ed alla riparazione di modelli tridimensionali esistenti. Sebbene la versione attualmente disponibile di OM non sia ancora quella definitiva, in quanto non tutte le sue parti risultano implementate nella loro completezza, tale tool sta già venendo impiegato con successo all'interno del gruppo VCG, ormai da tempo attivamente impegnato nella ricerca nel settore del 3D scanning. L'accuratezza e la precisione dei modelli con generati con OM nonché l'efficienza dimostrata nella loro creazione fanno pensare a OM come l'immediato successore di PlyMC2, il consolidato strumento di fusione fino ad oggi impiegato come standard nel gruppo VCG. In questo capitolo confronteremo i risultati generati da questi due strumenti a partire da dataset estremamente diversi fra loro sotto svariati aspetti, così da offrire un'analisi la più significativa possibile sulle potenzialità di OM. Sottolineiamo che tutti i test che riporteremo nel seguito sono stati realizzati con una versione non definitiva di OM che ancora non sfrutta il meccanismo out-of-core delineato in 3.4. Tale estensione è ancora in fase di sviluppo e l'implementazione ad oggi fornita non risulta ancora sufficientemente stabile per essere impiegata nella generazione dei seguenti test. Ad ogni modo i risultati e le prestazioni registrate da OM sono state decisamente soddisfacenti, nonostante l'intero

dataset dovesse risiedere completamente in memoria principale.

5.1 Octree Merger

Octree Merger è il tool di integrazione range map che abbiamo sviluppato durante la tesi a partire dai metodi e dalle tecniche descritti nei capitoli precedenti. Sviluppato interamente in C++ utilizzando Microsoft Visual Studio 2003 come ambiente di sviluppo, OM si articola in più di 6K linee di codice, che abbiamo compilato con successo anche col compilatore GNU g++. Di OM abbiamo realizzato due versioni differenti, una command line ed una con interfaccia grafica: quest'ultima, al fine di realizzare un'applicazione maggiormente portabile, è stata sviluppata utilizzando le class library di Qt, un framework per lo sviluppo di applicazioni cross-platform. Sebbene al lato pratico la versione maggiormente utilizzata sia quella a linea di comando, la versione con interfaccia grafica si è rivelata di determinante importanza durante le fasi di sviluppo e di debugging. In figura 5.1 uno screenshot della versione di OM con interfaccia grafica con caricato il dataset testa della Minerva di Arezzo utilizzato per i test di questo capitolo. La qualità associata ad un punto è codificata dal colore con cui questo è visualizzato: i punti con maggior grado di affidabilità sono colorati in blu, in rosso invece quelli di qualità inferiore. In figura 5.2 un'altro screenshot della versione con interfaccia grafica che mostra un particolare dello stesso dataset ed il rispettivo octree a livello di profondità sette. In figura 5.3 è visibile uno strumento che abbiamo scoperto essere di indubbia utilità per l'analisi del dataset: nella finestra in basso a sinistra dell'interfaccia è visualizzata l'intersezione del piano colorato in viola con le celle dell'octree; zoomando su questo strumento (figura 5.4) è possibile controllare la distribuzione dei punti rispetto alle foglie dell'octree ed avere un'immediata visione di eventuali errori di allineamento.

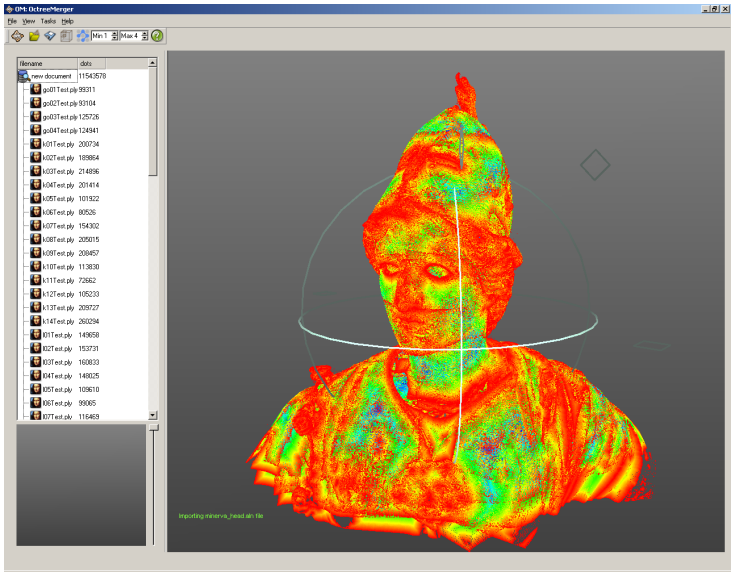


Figura 5.1: *Uno screenshot di OM con illustrato il dataset testa della Minerva di Arezzo utilizzato per questi test.*

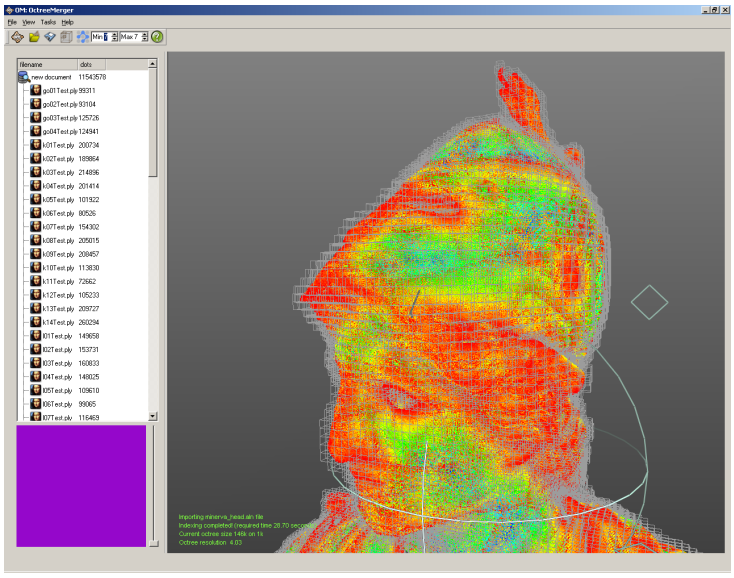


Figura 5.2: *Testa della Minerva di Arezzo con l'octree visualizzato solamente a profondità sette.*

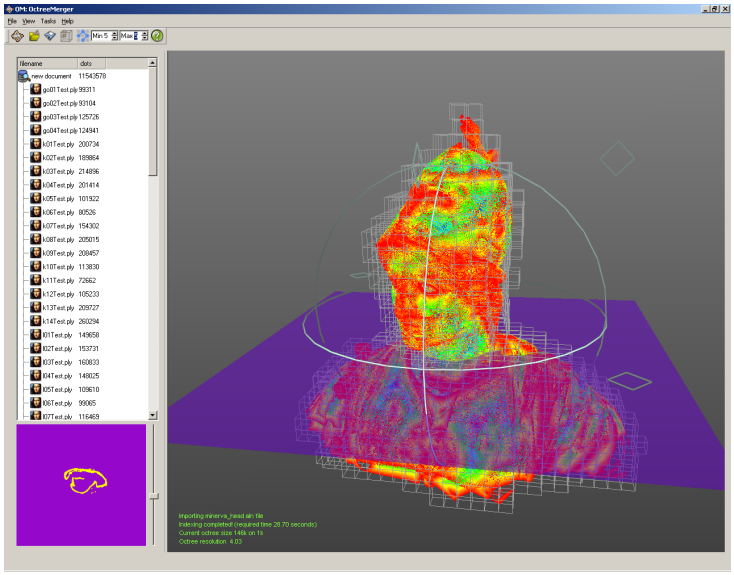


Figura 5.3: *Il piano in viola attraversa il dataset e l'octree sottostante, di cui è visualizzato solamente il livello cinque. In basso a sinistra è visualizzato l'intersezione del piano con le foglie dell'octree.*

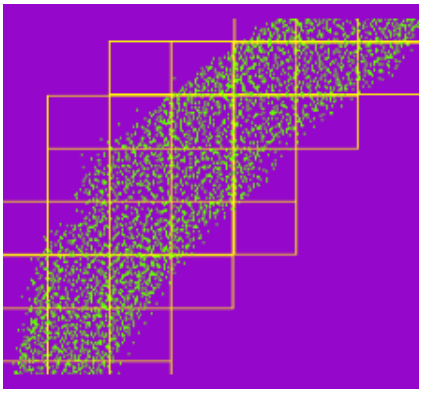


Figura 5.4: *Zoom sulla slice visualizzata nella figura precedente su una porzione del collo. I quadrati in giallo rappresentano le celle dell'octree, mentre in verde i punti del dataset contenuti al loro interno.*

Figura 5.5: *Capo tribù.*

Figura 5.6: *Minerva di Arezzo.*

5.2 I dataset di prova

La maggior parte dei dataset che abbiamo scelto di impiegare in questa sezione per confrontare OM e PlyMC2 sono stati acquisiti per mezzo di scanner a triangolazione, i quali consentono di ottenere range map molto dense con errore di campionamento generalmente molto basso. I soggetti di cui abbiamo generato i modelli tridimensionali sono "Capo tribù" (fig. 5.5), un bronzetto nuragico dell'età del bronzo (museo archeologico di Cagliari), il busto de "la Minerva di Arezzo" (fig. 5.6), bronzo del IV secolo D.C. (museo archeologico di Firenze) e "busto di Ippolita Sforza", l'unico calco in gesso di un'opera rinascimentale di Francesco Laurana (fig 5.7). Per tali modelli, le range map sono state acquisite con lo scanner a triangolazione Minolta VI900 in uso presso il gruppo VCG. A prova della stabilità del tool sviluppato a fronte di maggior rumore nel dataset di input, abbiamo voluto includere anche un dataset acquisito con scanner a tempo di volo: il modello, che abbiamo scelto per la ricchezza di particolari, è il rosone della cattedrale di Lucera Troia, in provincia di Foggia, le cui scansioni sono state effettuate attraverso lo scanner

Figura 5.7: *Busto di Ippolita Sforza.*

Figura 5.8: *Cattedrale di Troia.*

Leica Cyra 2500. Infine, per testare anche la capacità di OM di preservare le sharp feature presenti nel modello iniziale, abbiamo incluso anche un modello sintetico (il waved tetrahedron) generato ad hoc per questo particolare test. Le dimensioni dei dataset impiegati ed il numero delle rispettive range map sono riportati in tabella 5.1.

Soggetto	Range map	Campioni	Dimensione
busto di Ippolita Sforza	29	5 265 736	191 MB
capo tribù	102	9 864 132	756 MB
busto della Minerva di Arezzo	80	11 543 578	1.13 GB
rosone della cattedrale di Troia	8	6 327 762	320 MB
waved tetrahedron	1	524 290	25 MB

Tabella 5.1: *I dataset utilizzati per confrontare OM con PlyMC2.*

5.3 Prestazioni e risultati

Analizzeremo ora in questa sezione i modelli ricostruiti da PlyMC2 e da OM per i dataset di prova appena descritti. Per ogni dataset, abbiamo estratto la superficie a differenti livelli di risoluzione, cercando inoltre di utilizzare voxel di pari dimensioni sia per PlyMC2 sia per OM. Per ognuno di questi test verranno riportati i tempi di esecuzione dei due tool di fusione e la loro occupazione in memoria principale, espressa come dimensioni in MB del working set del processo e dello spazio virtuale di indirizzamento; tali misure sono riportate sulle ordinate dei successivi grafici, mentre nelle ascisse è riportato il tempo. Per tutti i modelli generati durante questi test verranno fornite le informazioni ricavate dall'analisi della mesh attraverso il tool TriMeshInfo, anch'esso sviluppato all'interno del gruppo VCG: tale strumento è stato progettato per l'analisi di modelli tridimensionali e per il recupero delle relative informazioni topologiche. Tutti i seguenti test sono stati effettuati su un PC con processore Intel Pentium IV dotato di 2GB di RAM e con sistema operativo Microsoft Windows 2000.

5.3.1 Busto di Ippolita Sforza

I test su questo dataset hanno evidenziato una maggiore sensibilità di OM rispetto a PlyMC2 circa gli errori di disallineamento: laddove infatti le range map non sono risultate perfettamente allineate, la superficie ricostruita da OM utilizzando i parametri di default presentava delle increspature e, in generale, una triangolarizzazione non ottimale(figura 5.9 a sinistra); viceversa, la ricostruzione effettuata da PlyMC2 non ha esibito tali problemi (figura 5.9 a destra). Per riuscire a superare anche con OM gli errori di disallineamento, abbiamo dovuto impiegare dei parametri diversi da quelli standard (figura 5.9 al centro): questa modifica ha però inciso sui tempi di esecuzione, che sono quasi raddoppiati. Per quanto concerne questi ultimi, abbiamo potuto

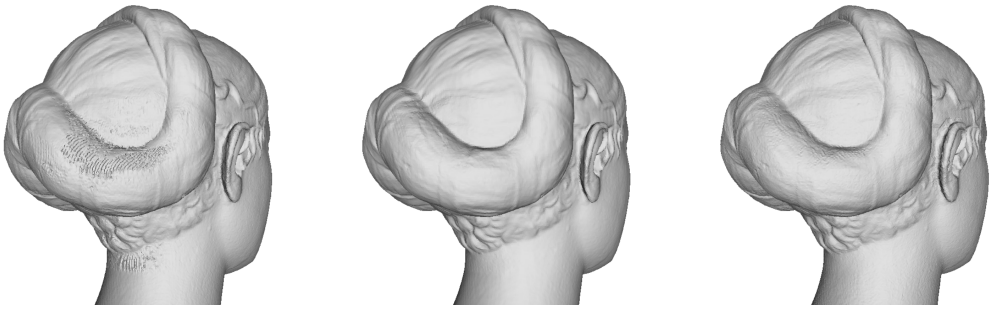


Figura 5.9: *Particolare della superficie generata da OM (sinistra) e da PlyMC2 (destra) per il dataset "busto di Ippolita Sforza" per la massima risoluzione con cui entrambi i tool hanno superato con successo il test, vale a dire con voxel cubici di 1.14 mm: le increspature sono causate da un errore di disallineamento delle range map. Al centro la superficie generata da OM utilizzando parametri di estrazione diversi da quelli di default.*

riscontrare una leggera differenza fra i tempi impiegati dai due metodi per la ricostruzione della superficie ai vari livelli di risoluzione: in tutti i test eseguiti, sebbene i modelli generati siano paragonabili in termini di numero di vertici e di facce, OM ha impiegato mediamente meno tempo rispetto a PlyMC2 (tabella 5.2). Facciamo notare infine che nessuno dei due algoritmi è riuscito a completare con successo la ricostruzione della superficie per la risoluzione più fine impostata a causa del superamento del limite della memoria.

		vertici			componenti		self	tempo
voxel	metodo	duplicati	tot.	facce	connesse	manifold	intersection	(mm:ss)
9.6 mm	OM	0	8 554	16 263	6	•	○	1:13
	PlyMC2	1	10 026	19 440	17	•	•	1:33
4.6 mm	OM	0	33 041	64 359	9	•	○	1:11
	PlyMC2	1	39 370	76 902	34	•	•	1:34
2.3 mm	OM	0	132 562	261 738	36	•	○	1:31
	PlyMC2	1	147 449	291 452	24	•	•	2:23
1.14 mm	OM	8	545 726	1 085 808	397	•	○	3:00
	PlyMC2	11	585 137	1 163 249	22	•	○	5:42
0.56 mm	OM							
	PlyMC2							

Tabella 5.2: *Esito dei test sul dataset busto di Ippolita Sforza; i tempi riportati per OM sono relativi all'utilizzo dei parametri di default.*

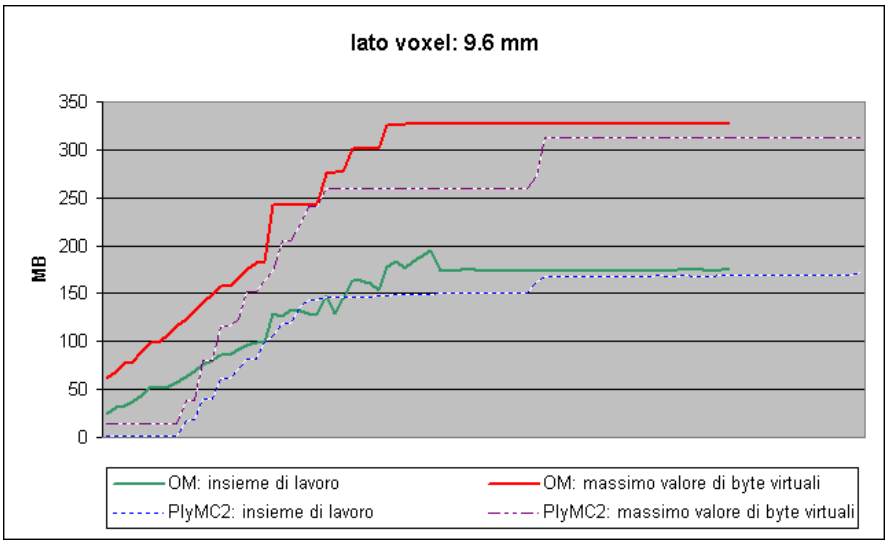


Figura 5.10: *Test su busto di Ippolita Sforza con voxel cubici di lato 9.6 mm.*

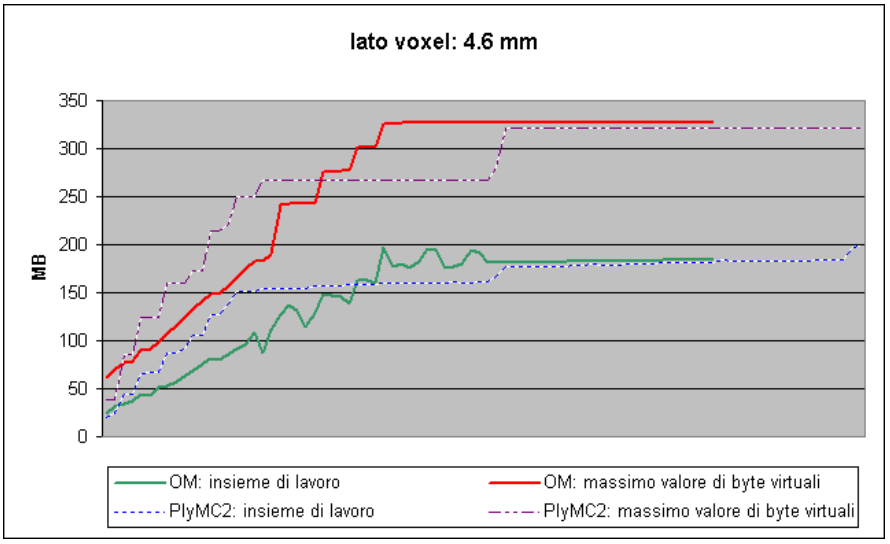


Figura 5.11: *Test su busto di Ippolita Sforza con voxel cubici di lato 4.6 mm.*

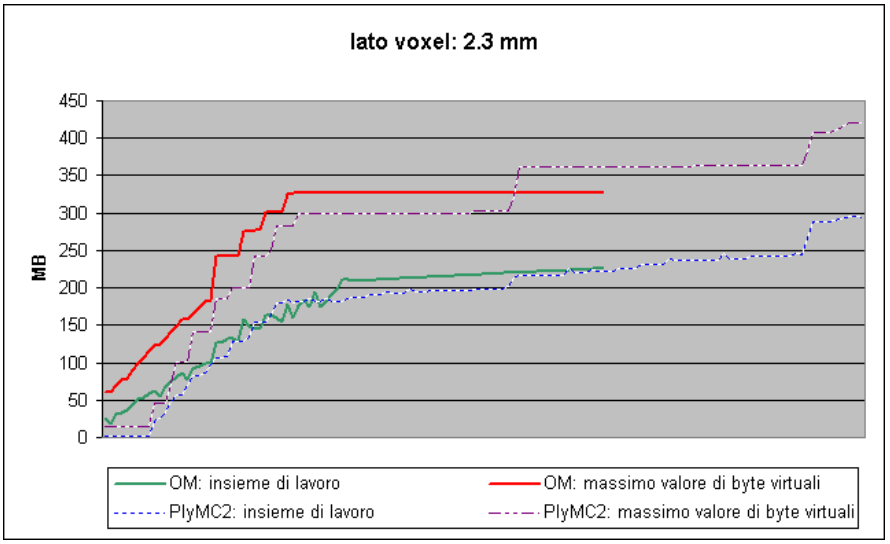


Figura 5.12: *Test su busto di Ippolita Sforza con voxel cubici di lato 2.3 mm.*

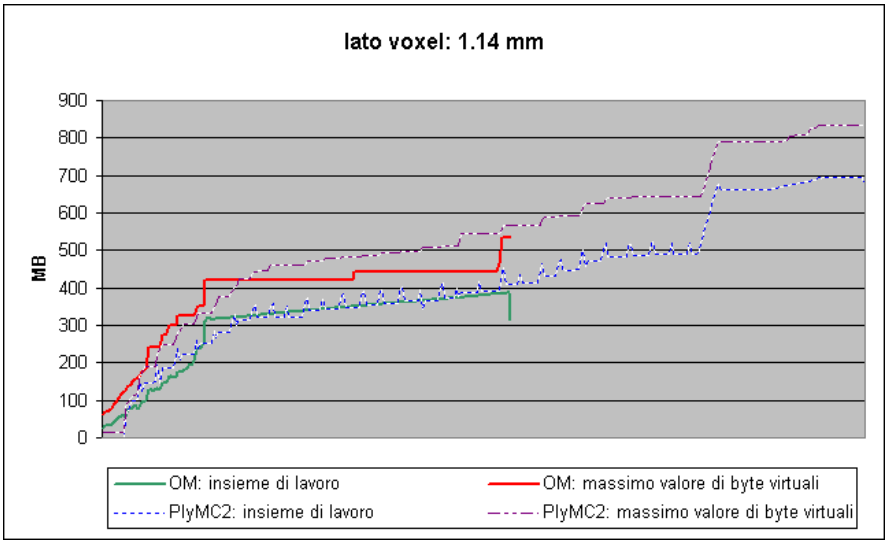


Figura 5.13: *Test su busto di Ippolita Sforza con voxel cubici di lato 1.14 mm.*

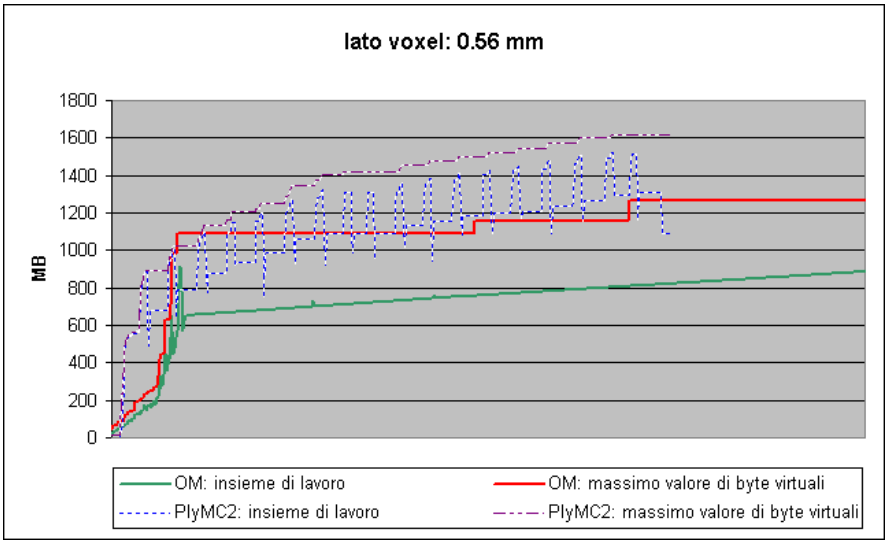


Figura 5.14: *Test su busto di Ippolita Sforza con voxel cubici di lato 0.56 mm.*

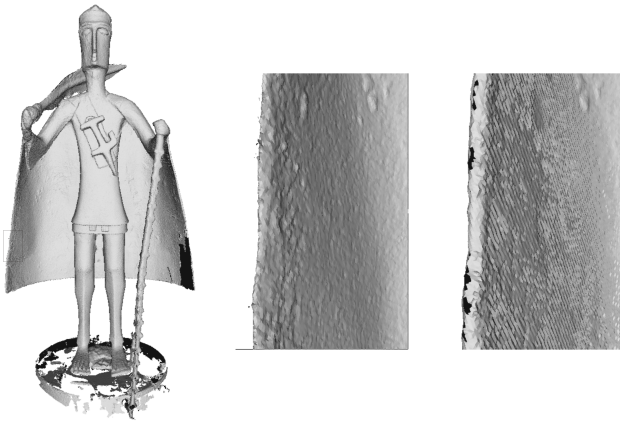


Figura 5.15: *A sinistra, il modello generato da OM; al centro e sulla destra un ingrandimento della porzione di superficie racchiusa dal rettangolo sui modelli generati rispettivamente da OM e da PlyMC2.*

5.3.2 Capo tribù

Entrambi i tool hanno completato con successo la ricostruzione della superficie a tutti i livelli di risoluzione richiesti durante il test. Tuttavia il modello generato da PlyMC2 presenta buchi ed increspature in porzioni di superficie che dovrebbero invece essere regolare: in figura 5.15 sulla destra un particolare preso dal mantello, mentre al centro la stessa porzione di superficie ricostruita correttamente da OM. Anche in questo caso i tempi impiegati da OM sono a parità di risoluzione mediamente inferiori a quelli di PlyMC2, con la sola eccezione della ricostruzione con voxel cubici di lato 0.93 mm (vedi tabella 5.3). Tuttavia, come evidenziato dai grafici 5.17 e 5.18, l'occupazione di memoria di OM per processare il volume su una griglia di risoluzione 1.87 mm. e 0.93 mm è stata decisamente maggiore rispetto a quella di PlyMC2: la differenza va imputata sicuramente allo spazio necessario a OM per mantenere in memoria l'indicizzazione dell'intero dataset ed il modello in fase di costruzione. Com'è infatti evidenziato dal grafico 5.20, anche PlyMC2 alla

voxel	metodo	vertici duplicati	vertici tot.	facce	componenti connesse	manifold	self intersection	tempo (mm:ss)
7.84 mm	OM	0	2 930	4 752	78	•	○	3:04
	PlyMC2	1	7 242	12 913	62	•	•	3:50
3.80 mm	OM	0	15 837	29 057	104	•	•	3:20
	PlyMC2	1	22 473	41 318	91	•	•	3:29
1.87 mm	OM	2	69 062	133 312	147	•	○	4:00
	PlyMC2	0	72 473	140 063	84	•	○	3:57
0.93 mm	OM	10	53 0835	1 037 602	256	•	○	8:08
	PlyMC2	15	84 0339	1 658 397	199	•	○	5:59
0.46 mm	OM	38	105 3223	2 067 672	467	•	○	10:00
	PlyMC2	35	102 9142	2 032 143	224	•	○	16:10

Tabella 5.3: *Esito dei test sul dataset Capo tribù.*

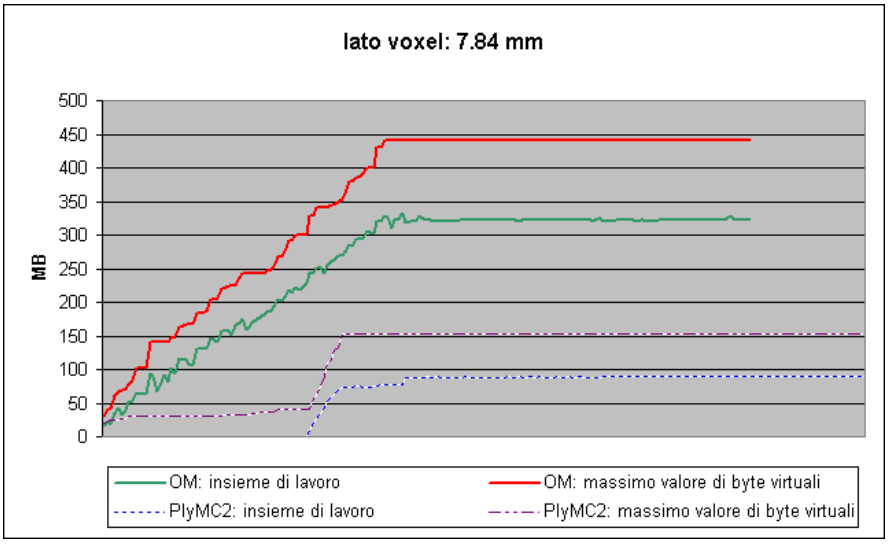


Figura 5.16: *Test su Capo tribù con voxel cubici di lato 7.84 mm.*

fine dell'analisi del volume ha un picco nell'occupazione di memoria, dovuto alla creazione del modello tridimensionale, il quale viene invece mantenuto costantemente in memoria da OM e aggiornato progressivamente durante la visita del volume.

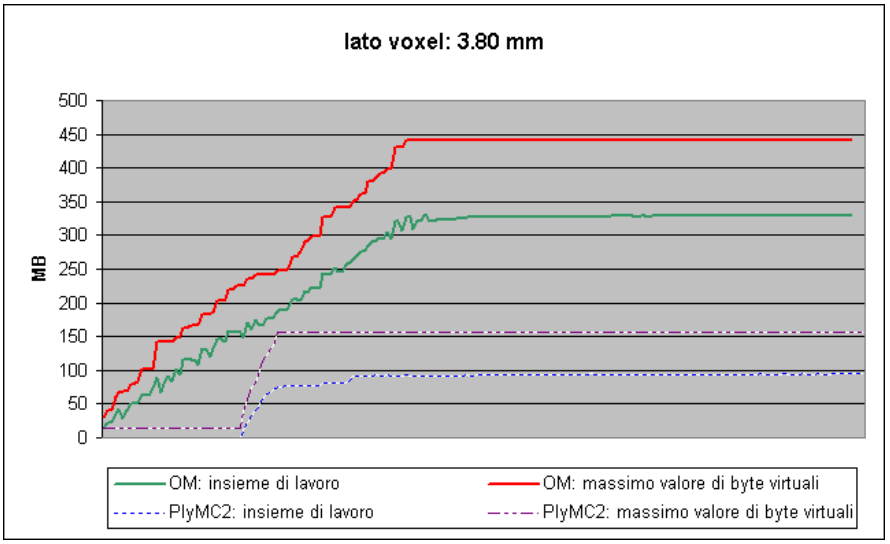


Figura 5.17: *Test su Capo tribù con voxel cubici di lato 3.80 mm.*

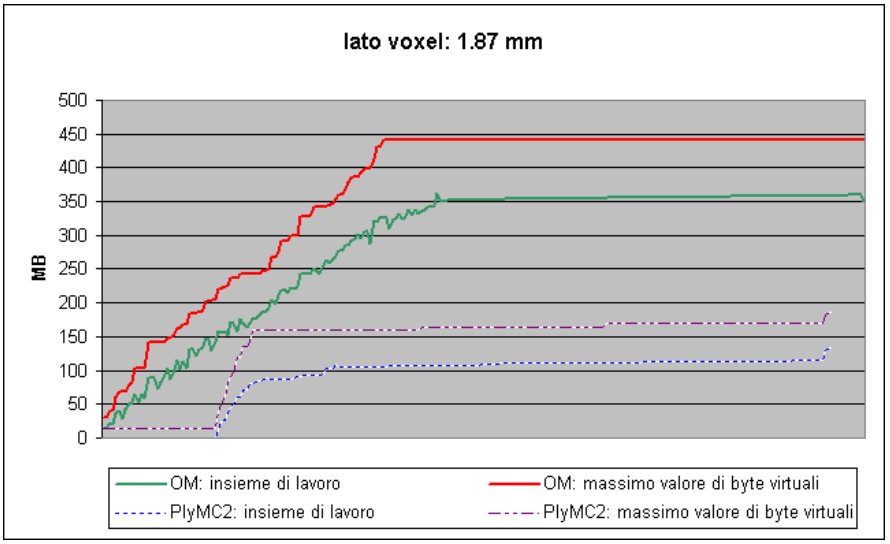


Figura 5.18: *Test su Capo tribù con voxel cubici di lato 1.87 mm.*

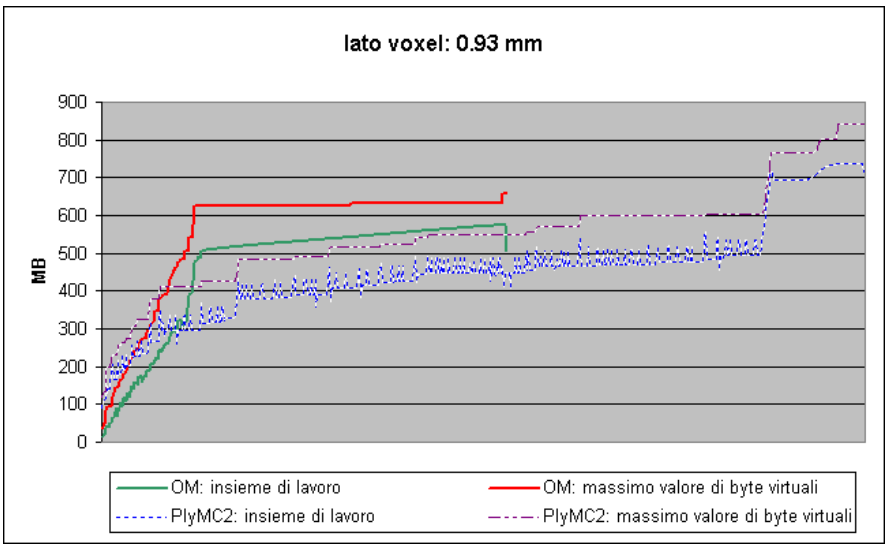


Figura 5.19: *Test su Capo tribù con voxel cubici di lato 0.93 mm.*

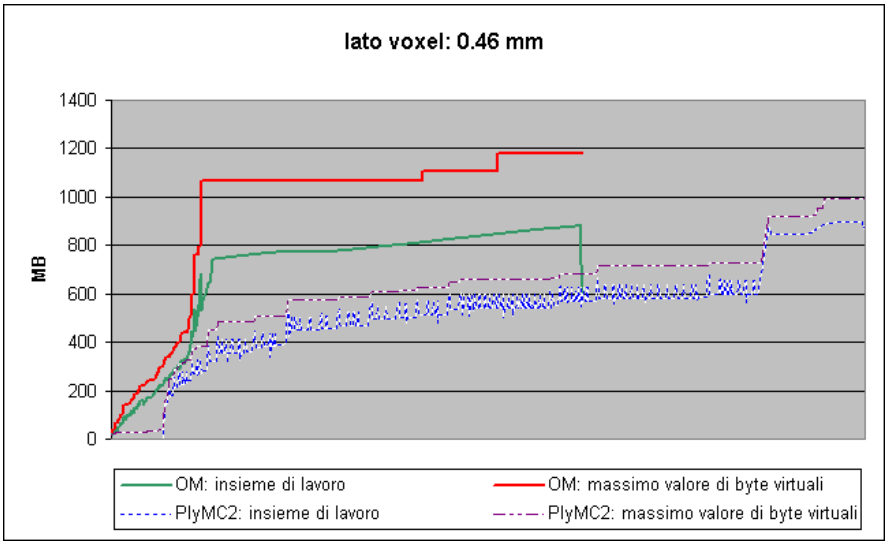


Figura 5.20: *Test su Capo tribù con voxel cubici di lato 0.46 mm.*

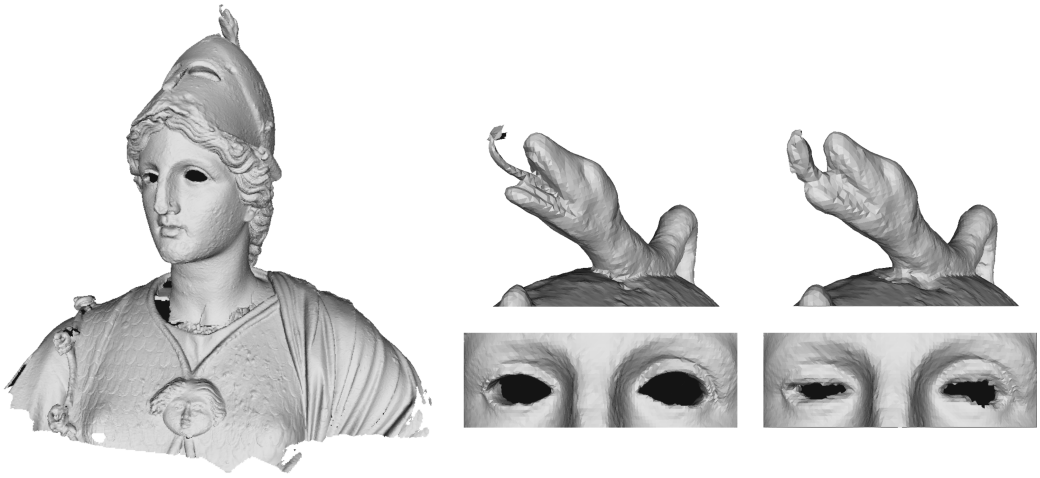


Figura 5.21: *A sinistra il modello generato da OM; al centro e a destra sono riportate le superfici ricostruite rispettivamente da OM e da PlyMC2 per gli occhi e per un dettagli sull'elmo.*

5.3.3 Testa della Minerva di Arezzo

Fra tutti i dataset che abbiamo preso in considerazione per queste prove la testa della Minerva di Arezzo è quello che per dimensioni (vedi tabella 5.1) avrebbe dovuto mettere maggiormente alla prova il nostro algoritmo. Com'era plausibile data l'assenza dell'estensione out-of-core, OM non è riuscito a ricostruire la superficie a risoluzione maggiore, cioè a 0.57 mm, terminando prematuramente durante la visita del volume. L'occupazione di memoria è stata decisamente maggiore rispetto a quella di PlyMC2 anche per la visita del volume ai livelli inferiori di risoluzione (vedi grafici 5.22, 5.23, 5.24 e 5.25): ad ogni modo, in tutti questi casi il volume è stato analizzato correttamente da OM, che ha sempre impiegato per l'estrazione della superficie un tempo inferiore a quello di PlyMC2. Inoltre, come anche riportato in figura 5.21, la superficie generata con OM è stata molto più accurata a parità di risoluzione di quella generata da PlyMC2.

		vertici			componenti		self	tempo
voxel	metodo	duplicati	tot.	facce	connesse	manifold	intersection	(mm:ss)
9.74 mm	OM	0	6 525	11 836	24	•	•	3:20
	PlyMC2	1	7 031	13 700	7	•	•	4:08
4.72 mm	OM	0	33 177	64 209	19	•	○	3:32
	PlyMC2	1	28 066	55 065	24	•	•	4:23
2.32 mm	OM	1	131 800	25 8422	26	•	•	5:13
	PlyMC2	1	109 345	21 5388	38	•	•	5:19
1.15 mm	OM	7	514 322	1 015 070	83	•	•	7:37
	PlyMC2	8	434 631	859 334	63	•	○	9:36
0.57 mm	OM							
	PlyMC2	44	1 709 949	3 393 479	76	•	○	29:28

Tabella 5.4: *Esito dei test sul dataset testa della Minerva di Arezzo.*

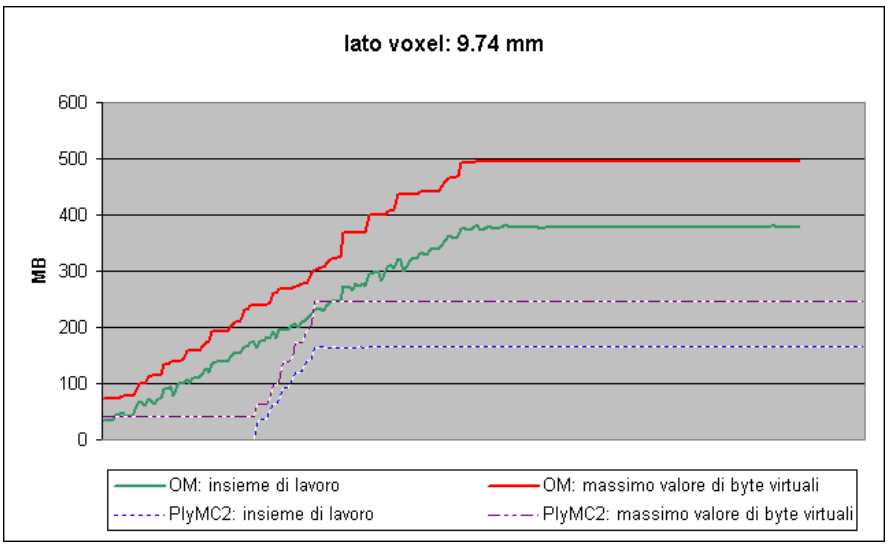


Figura 5.22: *Test sulla testa della Minerva di Arezzo con voxel cubici di lato 9.74 mm.*

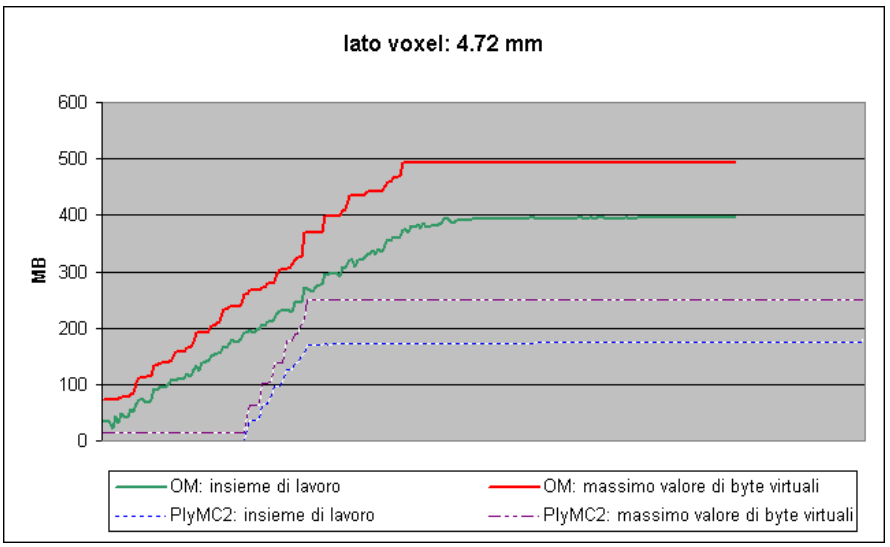


Figura 5.23: *Test sulla testa della Minerva di Arezzo con voxel cubici di lato 4.72 mm.*

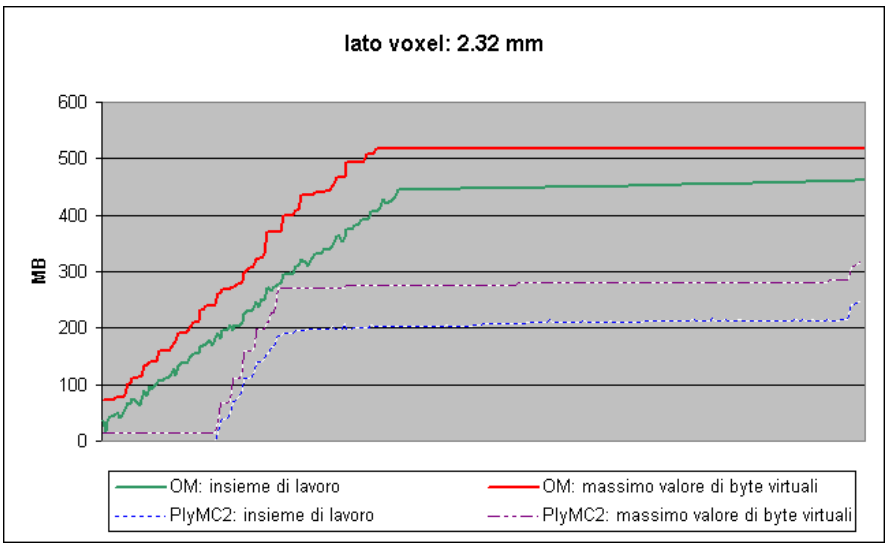


Figura 5.24: *Test sulla testa della Minerva di Arezzo con voxel cubici di lato 2.32 mm.*

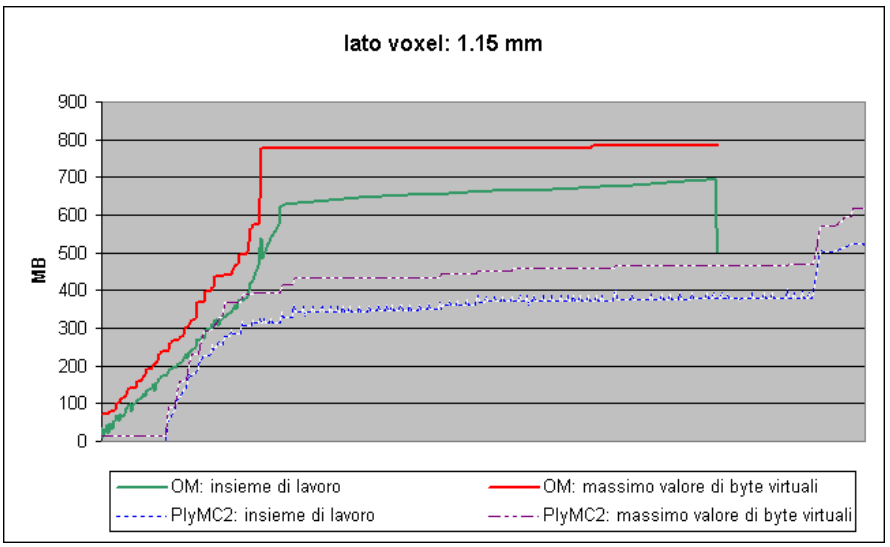


Figura 5.25: *Test sulla testa della Minerva di Arezzo con voxel cubici di lato 1.15 mm.*

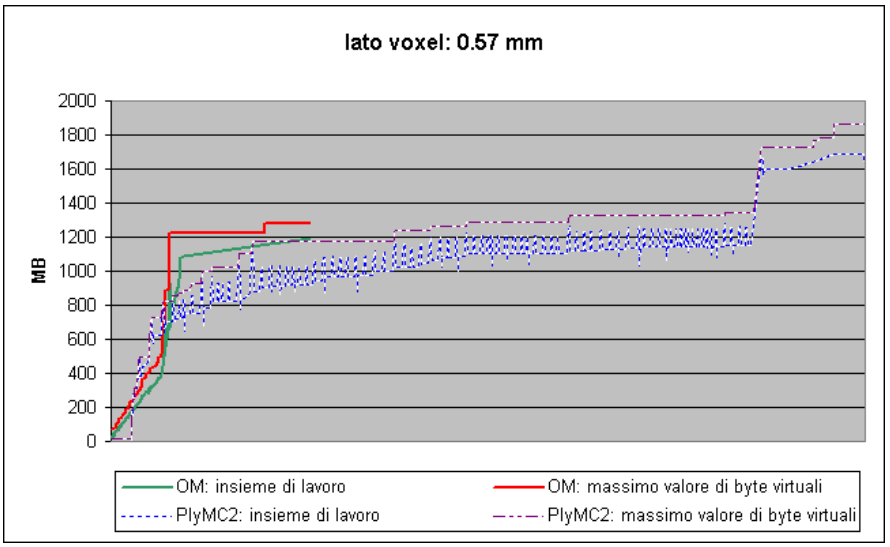


Figura 5.26: *Test sulla testa della Minerva di Arezzo con voxel cubici di lato 0.57 mm.*

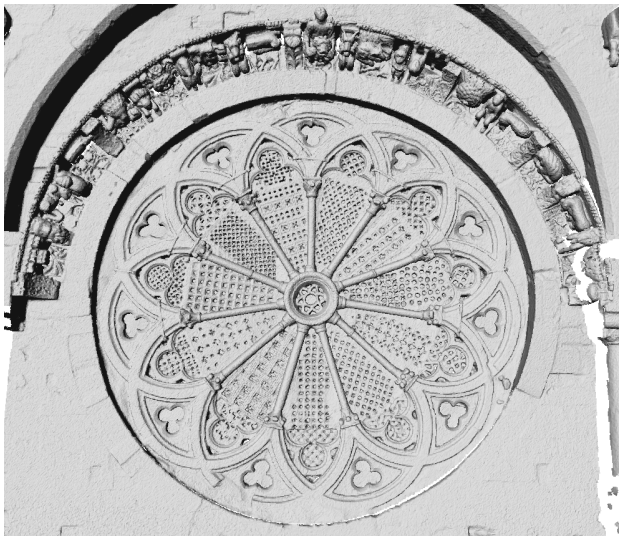


Figura 5.27: *Il rosone nel modello generato da OM a massima risoluzione per il dataset cattedrale di Lucera Troia.*

5.3.4 Rosone della cattedrale di Lucera Troia

Come accennato in precedenza, questo dataset è l'unico fra quelli adottati per questi test ad essere stato acquisito con uno scanner a tempo di volo: di conseguenza l'errore presente in questo campionamento è percentualmente maggiore rispetto a quello presente negli altri dataset. Sia OM che PlyMC2 hanno saputo gestire correttamente il dataset e sono riusciti ad estrarre la superficie a diversi livelli di risoluzione; su questo particolare dataset OM è riuscito anche, a differenza di PlyMC2, a ricostruire la superficie al minimo livello di dettaglio richiesto, vale a dire a 1.43 cm. (figura 5.27). Indipendentemente dal livello di dettaglio utilizzato per la ricostruzione della superficie, OM non solo ha richiesto meno tempo per processare l'intero volume, ma ha anche richiesto molta meno memoria rispetto a PlyMC2 (figure 5.28-5.32 e tabella 5.5). Questo risultato è a nostro avviso dovuto alla particolare distribuzione del dataset rispetto al volume: poiché infatti i campioni si tro-

		vertici			componenti		self	tempo
voxel	metodo	duplicati	tot.	facce	connesse	manifold	intersection	(mm:ss)
24 cm	OM	0	5 862	10 974	6	•	•	1:50
	PlyMC2	1	4 977	9 394	1	•	•	2:12
11 cm	OM	0	22 587	42 946	23	•	•	1:39
	PlyMC2	0	19 604	37 050	23	•	•	2:28
5.7 cm	OM	2	90 858	174 820	101	•	•	1:50
	PlyMC2	1	91 022	174 618	58	•	•	2:40
2.27 cm	OM	13	354 413	687 966	148	•	○	3:15
	PlyMC2	3	358 088	692 588	116	•	○	3:48
1.43 cm	OM	103	1 359 602	2 641 789	213	•	○	6.41
	PlyMC2							

Tabella 5.5: *Esito dei test sul dataset Rosone della cattedrale di Troia.*

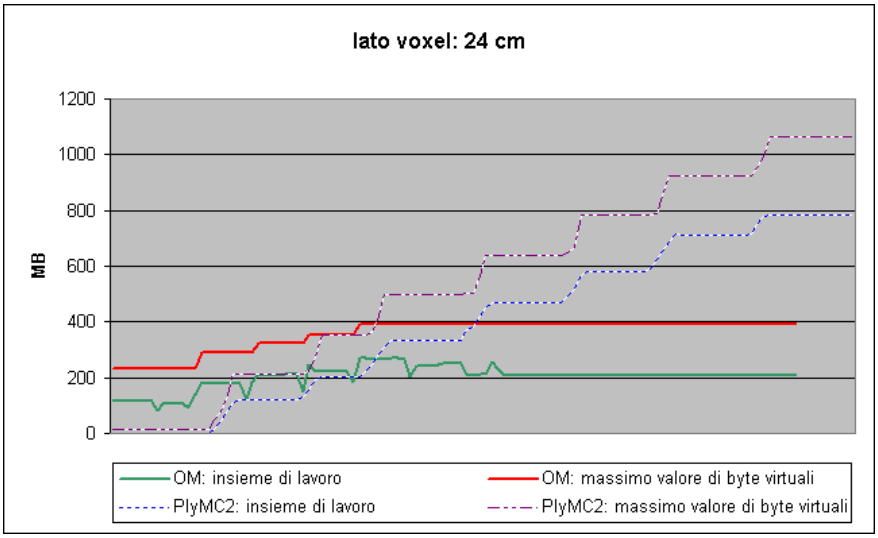


Figura 5.28: *Test su Rosone della cattedrale di Troia: voxel cubici di lato 24 cm.*

vano all'incirca allineati su uno stesso piano, l'octree costruito da OM per l'indicizzazione del dataset è stato espanso solamente in una porzione di volume limitata, a differenza invece di quanto accadeva con i dataset visti in precedenza.

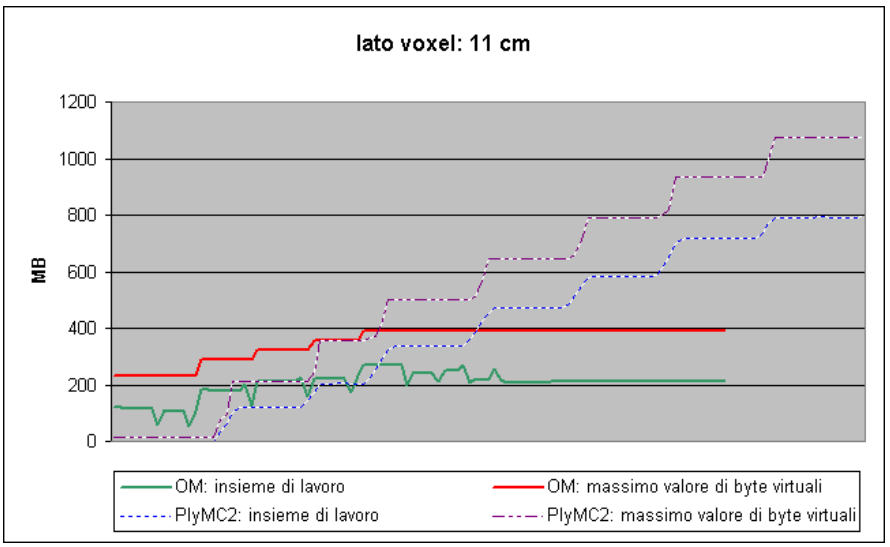


Figura 5.29: *Test su Rosone della cattedrale di Troia: voxel cubici di lato 11 cm.*

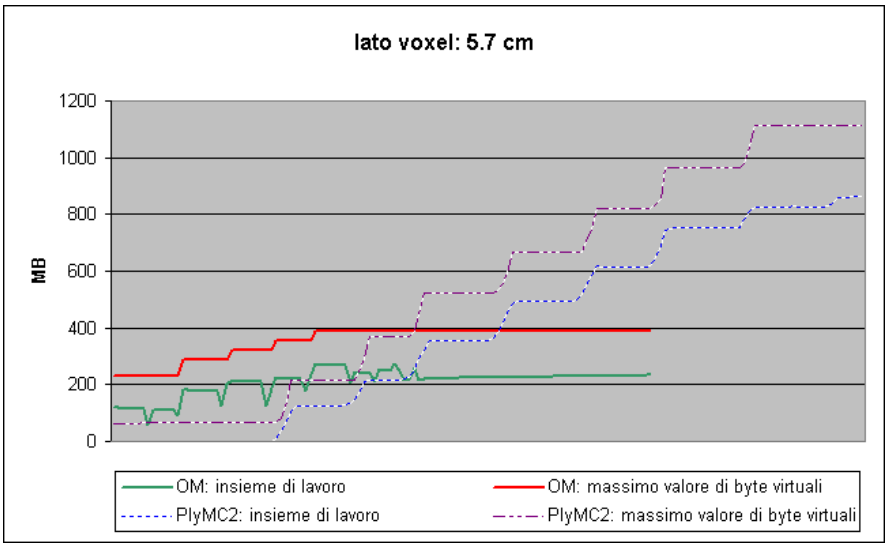


Figura 5.30: *Test su Rosone della cattedrale di Troia: voxel cubici di lato 5.7 cm.*

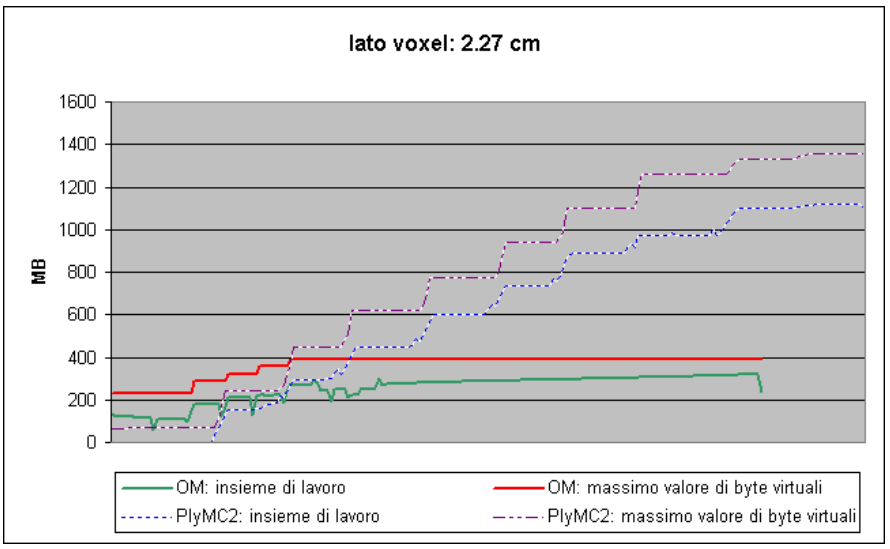


Figura 5.31: *Test su Rosone della cattedrale di Troia: voxel cubici di lato 2.27 cm.*

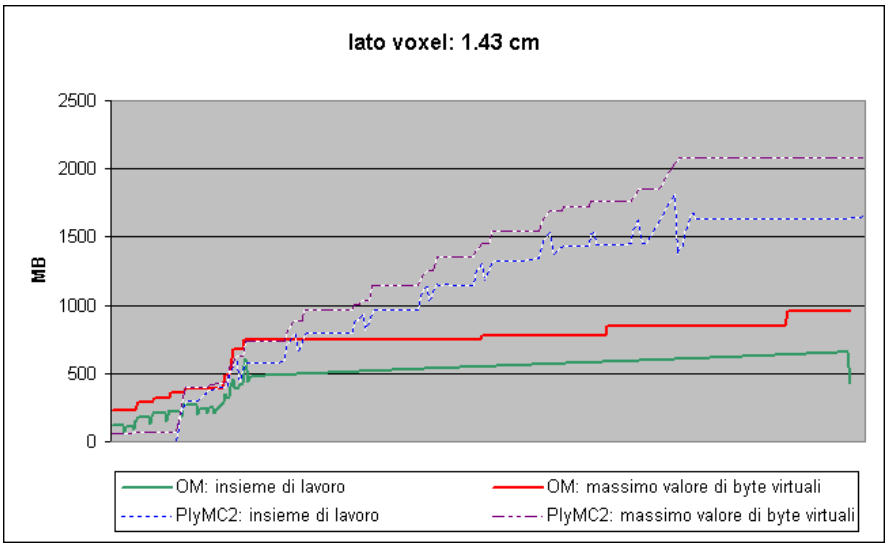


Figura 5.32: *Test su Rosone della cattedrale di Troia: voxel cubici di lato 1.43 cm.*

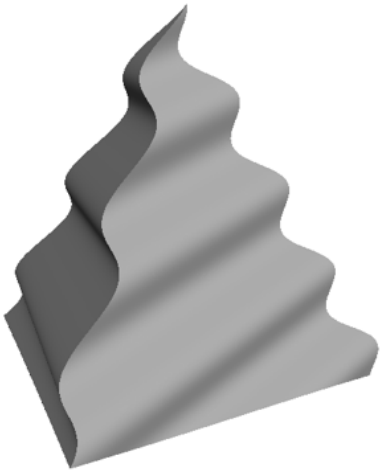


Figura 5.33: *Waved tetrahedron.*

Figura 5.34: *Architrave di S.Ranieri (particolare).*

5.3.5 Waved tetrahedron

Il waved tetrahedron, illustrato in figura 5.33, è un modello sintetico che abbiamo generato per mettere alla prova le capacità del nostro algoritmo di preservare eventuali sharp features presenti nel dataset in input. Differentemente dai precedenti test, la ricostruzione della superficie è stata effettuata solamente a tre diversi livelli di risoluzione: estrarre la superficie ad una risoluzione maggiore avrebbe significato infatti generare un modello più dettagliato di quello in input. Sia PlyMC2 che OM sono riusciti a generare i modelli per i vari livelli di risoluzione richiesti ed il divario tra i loro tempi di esecuzione è stato minimo (vedi tabella 5.6); in questo caso PlyMC2 ha richiesto anche una maggior quantità di memoria rispetto ad OM (figure 5.36-5.38). Per quanto riguarda invece lo scopo vero e proprio di questo test, abbiamo potuto riscontrare che OM offre maggiori garanzie rispetto a PlyMC2 sulla capacità di preservare le sharp feature presenti nel dataset (vedi figura 5.35). Ad ogni modo sottolineamo che in prossimità di spigoli la qualità della triangolarizzazione generata da OM è mediocre: questo pro-

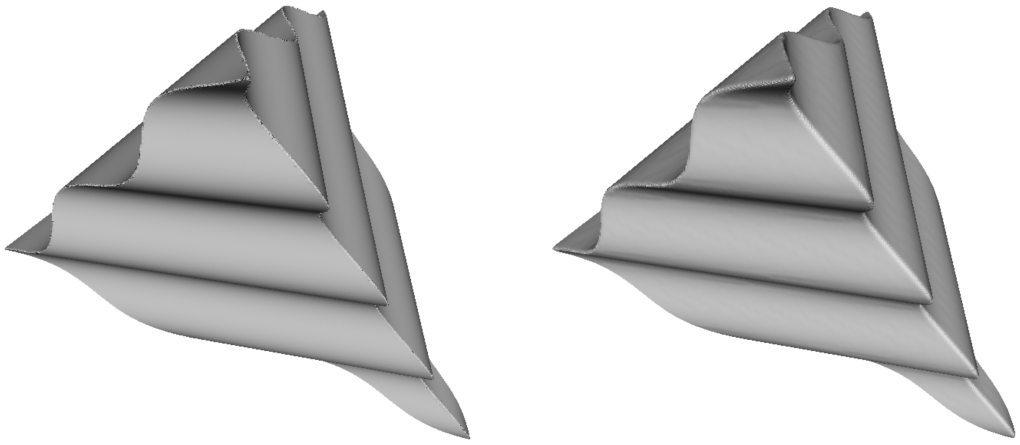


Figura 5.35: *Il modello ricostruito da OM (sinistra) le sharp feature sono più marcate rispetto al modello generato da PlyMC2 (destra).*

		vertici			componenti		self	tempo
voxel	metodo	duplicati	tot.	facce	connesse	manifold	intersection	(mm:ss)
$^1/_{32}$	OM	0	20 994	41 984	1	•	•	0:12
	PlyMC2	1	19 523	39 007	3	•	•	0:14
$^1/_{64}$	OM	0	88 111	176 218	1	•	•	0:18
	PlyMC2	3	83 119	166 219	2	•	•	0:22
$^1/_{128}$	OM	2	361 037	722 068	1	•	○	0:53
	PlyMC2	9	335 433	670 860	1	•	○	0.54

Tabella 5.6: *Esito dei test sul dataset waved tetrahedron.*

blema non è imputabile tanto all'algoritmo di poligonalizzazione utilizzato per questo test (l'extended marching cubes), ma alla tendenza del metodo di proiezione di punti di *smussare* la superficie in prossimità delle sharp features.

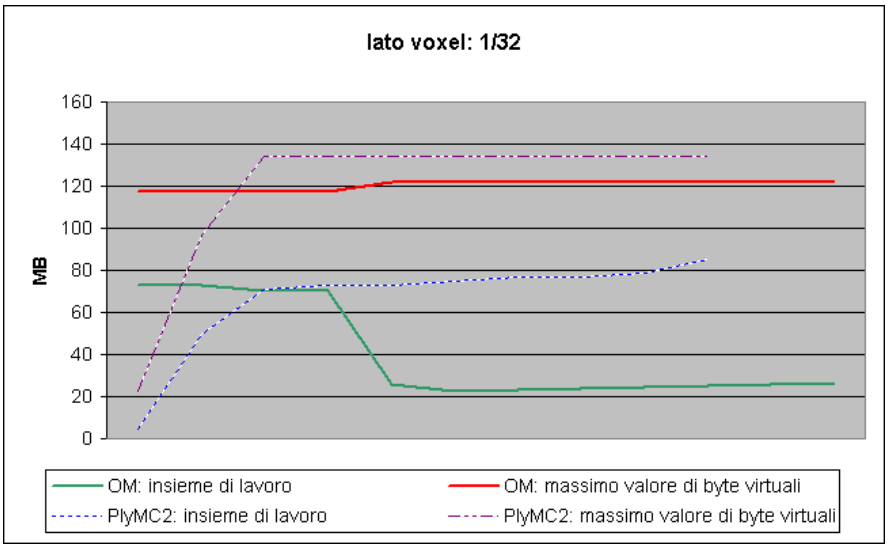


Figura 5.36: *Test sul waved tetrahedron con voxel cubici di lato* $^1/_{32}$ *dell'originale.*

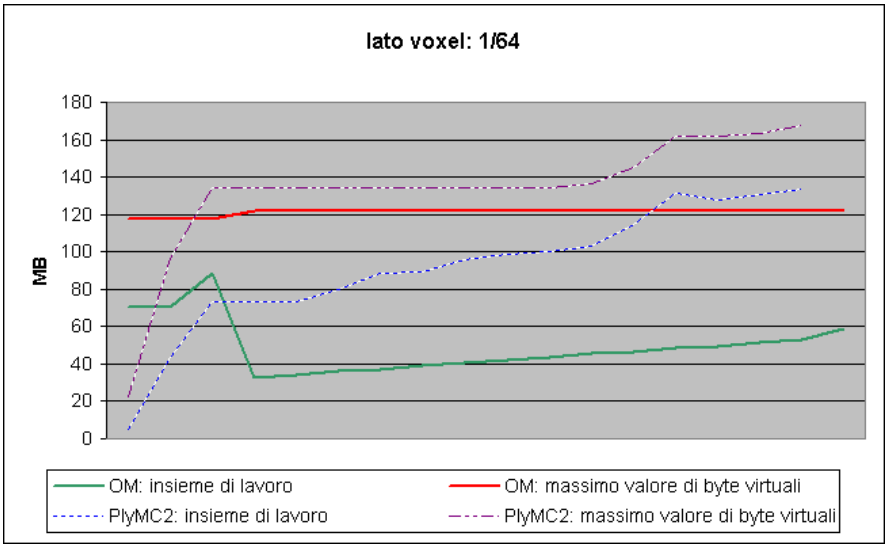


Figura 5.37: *Test sul waved tetrahedron con voxel cubici di lato* $^1/_{64}$ *dell'originale.*

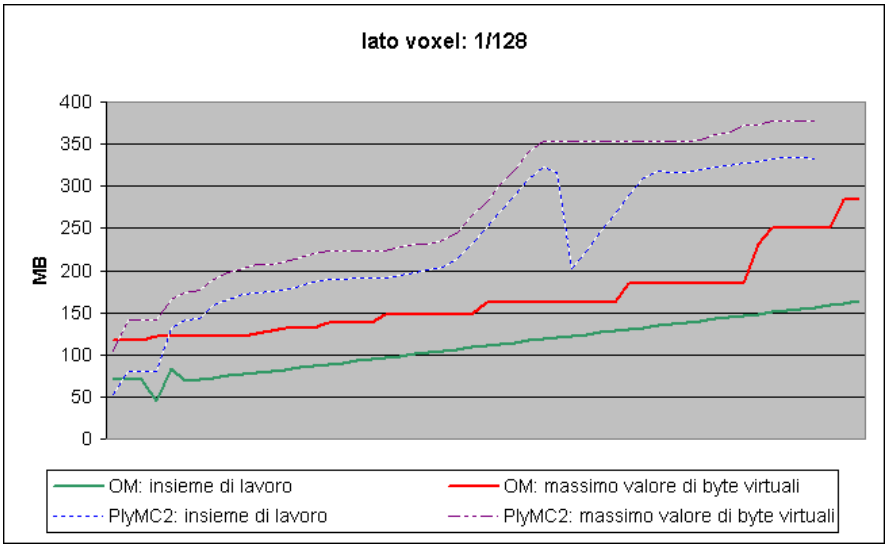


Figura 5.38: *Test sul waved tetrahedron con voxel cubici di lato* $^1/_{128}$ *dell'originale.*

5.4 L'influenza dei parametri

Concludiamo infine questo capitolo con una breve e riassuntiva descrizione dei parametri esposti all'utente nella fase di estrazione della superficie: per ognuno di essi forniremo una descrizione del suo significato ed illustreremo come esso si ripercuota sulla superficie ricostruita. A questo scopo abbiamo scelto un nuovo dataset, un particolare preso dall'architrave della porta della cattedrale di S. Ranieri in Pisa, illustrato in figura 5.34. Il dataset che utilizzeremo è di dimensioni ridotte (6M di punti), ma è estremamente utile per illustrare le capacità dell'algoritmo adottato in OM di tappare piccoli buchi dovuti alle occlusioni presenti nella superficie.

Tra i vari parametri quello sicuramente di più immediata comprensione è la dimensione k dell'insieme *Nbhd* (pagine 25 e 74) da utilizzare durante la procedura di proiezione dei punti: ovviamente più la cardinalità di questo insieme è grande, tanto maggiore la superficie ricostruita tenderà a perdere

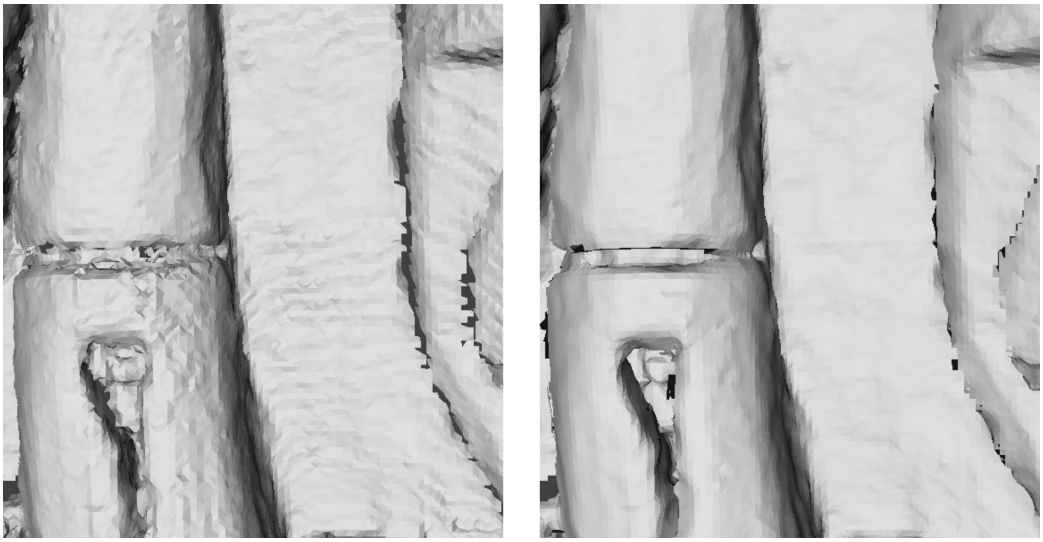


Figura 5.39: *Un valore maggiore del parametro k consente di generare delle superfici maggiormente smooth. A sinistra la superficie è stata generata con $k = 5$, a destra con $k = 64$.*

le caratteristiche presenti nel dataset originario. In figura 5.39 sono riportate due particolari delle superfici ricostruite cambiando solamente la cardinalità dell'insieme k-neighborhood e mantenendo costanti tutti gli altri parametri. Ovviamente la dimensione k dell'insieme $Nbhd$ si ripercuote sui tempi di calcolo necessari all'individuazione della superficie, in quanto ad ogni passo l'energy function viene valutata su un insieme di campioni di cardinalità esattamente k. Per superfici sufficientemente regolari, l'aumento dei tempi di calcolo derivante da un valore di k sufficientemente grande può essere bilanciato attraverso il parametro di tolleranza del metodo di Brent per la minimizzazione dell'energia: per valori del fattore di tolleranza molto vicini allo zero, la procedura di minimizzazione convergerà solamente per punti che con altissima probabilità appartengono alla superficie, e di conseguenza il tempo necessario alla loro individuazione sarà maggiore; aumentando invece

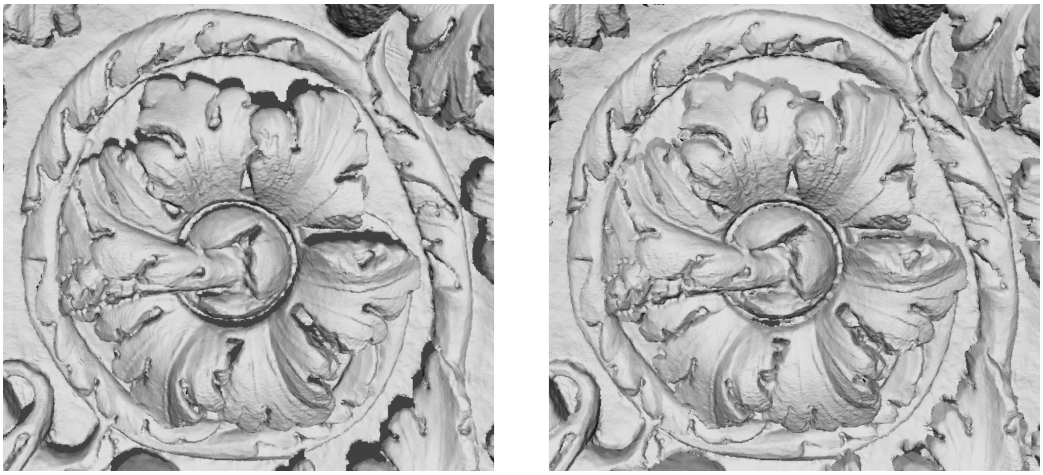


Figura 5.40: *Ricostruzione di un particolare dell'architrave della cattedrale di San Ranieri per differenti valori del numero di passi di espansione e del parametro hole-filling.*

tale parametro, la procedura di minimizzazione considererà come punti della superficie anche punti che in realtà risultano a lei solamente molto vicini, raggiungendo quindi con maggior velocità la convergenza.

Una delle interessanti proprietà del metodo da noi proposto è la capacità di riuscire ad estendere la superficie anche in quelle porzioni in cui, a causa delle occlusioni presenti nella superficie del modello, il campionamento del dataset risultasse non sufficiente. Il grado con cui la superficie viene estesa per tappare questi buchi è regolata attraverso due parametri, il numero di *passi di espansione* e l'*hole-filling level*. Come accennato in 4.2, il nostro algoritmo inizia a ricostruire la superficie a partire da quei voxel che sicuramente ne contengono una porzione, vale a dire i voxel in cui ricada almeno un punto del dataset. Molto probabilmente però la superficie attraverserà anche voxel adiacenti, nei quali potrebbe anche non ricadere alcun punto del dataset. I numero di passi di espansione è il parametro che permette di superare questo problema, imponendo all'algoritmo di estrazione di visitare anche

quei voxel che, pur non contenendo nessun campione, possono probabilmente essere attraversati dalla superficie. Abbiamo notato però che l'utilizzo di tale approccio senza alcuna contromisura può portare alla generazione nelle superficie ricostruita di artefatti non presenti nel modello di partenza, quali porzioni sconnesse di superficie. Data l'impossibilità di individuare dei vincoli validi in assoluto per contrastare questo problema, abbiamo ritenuto opportuno lasciare all'utente la libertà di impostare il livello con cui l'algoritmo estende la superficie sulle porzioni poco campionate. Il significato del parametro hole-filling level è legato alla distanza fra un punto $\boldsymbol{p}$ nello spazio ed il centro di massa dell'insieme $k\text{-}Nbhd(\boldsymbol{p})$: più precisamente un voxel marcato per la visita durante un passo di espansione viene effettivamente analizzato se la distanza fra ognuno dei suoi corner ed i centri di massa dei rispettivi insiemi $k\text{-}Nbhd$ risulta non maggiore di hole-filling volte la diagonale del voxel. Gli effetti sulla superficie ricostruita derivanti dal cambiamento di questi due parametri sono evidenti in figura 5.40: come è possibile notare l'algoritmo è stato capace di estendere ragionevolmente la superficie anche su quelle aree in cui a causa delle occlusioni del modello nel dataset mancavano completamente dei campioni. Ovviamente il cambiamento di questi due parametri ha delle ripercussioni sui tempi di esecuzione dell'algoritmo, in quanto aumentano notevolmente il numero di voxel visitati durante il processo di estrazione della superficie: a titolo di esempio, si consideri che il tempo richiesto per generare il modello sulla destra in figura 5.40 è stato circa tre volte il tempo necessario alla generazione del modello sulla sinistra.

Capitolo 6

Conclusioni

Scopo di questa tesi è stato fin dal principio la realizzazione di un tool da adoperare nella fase conclusiva della pipeline di acquisizione, vale a dire quella di integrazione: durante questa fase la nuvola di punti viene analizzata per ricavare il modello tridimensionale da essa descritto. I requisiti cui questo tool doveva rispondere sono stati delineati con chiarezza fin dal principio: dato le dimensioni dell'input, l'algoritmo doveva essere in grado di gestire efficientemente dataset comprendenti milioni di punti e di processare velocemente il volume da questo occupato. Inoltre era richiesto esplicitamente che la tecnica adottata dall'algoritmo per la descrizione del modello fosse basata su una rappresentazione implicita: questo vincolo è derivato dalle interessanti proprietà in termini di qualià e di accuratezza che i metodi impliciti riescono in generale a garantire.

Si è reso quindi necessario un'analisi comparativa dei vari metodi esistenti finalizzata all'individuazione dell'approccio che meglio si adattasse alle nostre esigenze. Lo studio ed il confronto di tali metodi ci ha permesso di individuare nell'approccio basato sui point-set un ottimo punto di partenza su cui impostare il nostro algoritmo di ricostruzione di superfici. Le garanzie offerte da questo approccio in termini di qualità della rappresentazione generata e la formulazione capace di gestire tutti i dati di nostro interesse (posizione, normale, colore e qualità) si sono infatti rivelate fin da subito estremamente

adatte agli obbiettivi preposti.

Prima di procedere all'effettiva implementazione del nuovo fonditore abbiamo ritenuto indispensabile realizzare innanzitutto un framework efficiente per la gestione del dataset, grazie al quale svincolare quanto più possibile il metodo di ricostruzione di superfici da dettagli di rappresentazione e di gestione dell'input. Il framework che abbiamo realizzato consente di importare sia dataset provenienti dal processo di acquisizione e di allineamento sia modelli tridimensionali già esistenti, così che possa essere utilizzato indistintamente per la generazione di un nuovo modello tridimensionale o per il *remeshing* di uno già esistente. La struttura portante dell'intero framework è costituita da un octree, una struttura dati ampiamente utilizzata per la organizzazione e la gestione di dataset volumetrici. Dato il vincolo che ci eravamo prefissato di basso overhead in termini di occupazione di memoria, abbiamo optato per una politica di espansione dell'octree adattiva, in modo tale da evitarne la costruzione in quelle porzioni del volume in cui non cade alcun punto del dataset; questa decisione rende possibile inoltre variare localmente la complessità del modello generato in funzione della complessità del soggetto di partenza. L'octree che abbiamo implementato tiene traccia per ogni nodo dell'insieme di punti contenuti all'interno del voxel associato a quest'ultimo. Al fine di minimizzare lo spazio necessario al mantenimento delle informazioni di indicizzazione, l'intera nuvola di punti viene mantenuta ordinata secondo la tecnica dello z-order, grazie alla quale l'ordinamento imposto ai punti rispetto alla massima altezza dell'albero si mantiene coerente ad un qualsiasi altro livello di risoluzione minore. Per via di quest'ultima proprietà, l'indicizzazione dell'intero dataset può essere completata assegnando ad ogni nodo dell'albero l'intervallo, rappresentato come coppia di indici, dei punti che cadono all'interno del voxel associatogli. La memorizzazione di questa informazione costituisce l'elemento chiave per garantire l'efficiente risoluzione di vari tipi di spatial query. Attualmente il nostro framework è capace di

rispondere efficientemente alle query spaziali più comuni, vale a dire le range query e il k-neighborhood: il risultato di ognuna di queste query può essere generato grazie alle informazioni di indicizzazione semplicemente attraverso una vista top-down dell'octree.

Sulla base di questo framework abbiamo poi implementato l'algoritmo di estrazione di superfici. Il maggior ostacolo che abbiamo incontrato durante il suo sviluppo ha riguardato l'individuazione di una procedura che consentisse di passare dalla rappresentazione a punti adottata dal metodo point-set, scelto come kernel del nostro algoritmo, ad una rappresentazione invece del tipo mesh simpliciale. La soluzione da noi delineata si basa su una reinterpretazione della procedura di proiezione dei punti sulla superficie che ci ha permesso di definire uno scalar field su tutto il volume occupato dal dataset. In particolare i punti sulla superficie individuati dalla procedura di proiezione e la direzione utilizzata per la loro generazione permettono di identificare un piano, il quale intuitivamente, costituisce la miglior approssimazione locale alla superficie. Grazie a tali piani, la distanza di un punto nello spazio dalla superficie coincide con la distanza del punto dal piano individuabile a partire dai risultati generati dalla procedura di proiezione a lui applicata. Combinando quindi lo scalar field così definito con la griglia individuata dai voxel dell'octree è possibile sfruttare un qualsiasi algoritmo di poligonalizzazione per generare infine una mesh simpliciale.

Il tool che abbiamo sviluppato su questi metodi ha permesso di processare correttamente dataset composti da milioni di punti e di estrarre la superficie implicata da questi ultimi in un tempo in media inferiore rispetto a quello richiesto da PlyMC2, il tool attualmente impiegato come fonditore all'interno del gruppo VCG presso cui questa tesi è stata svolta. Per di più il confronto fra i modelli generati dai due strumenti a parità di risoluzione ha quasi sempre evidenziato la capacità del nostro algoritmo di preservare quei dettagli e quelle caratteristiche presenti nel dataset che invece venivano persi nei model-

li generati con PlyMC2. Particolarmente evidenti sono risultate le differenze fra i modelli generati dalle scansioni effettuate allo scafo del FARR40, un prototipo di una barca da regata di 12 metri, un dataset che ha rimarcato la ottime potenzialità del nostro algoritmo: laddove infatti PlyMC2 incontrava grosse difficoltà dovute soprattutto alla presenza di campioni generati su entrambi i lati di una superficie molto sottile, OM invece riusciva con successo a generare una rappresentazione accurata e ricca di particolari.

6.1 Lavori futuri

La qualità dei modelli generati con OM nonché i tempi mediamente contenuti necessari alla loro creazione hanno permesso di identificare in questo tool l'immediato successore di PlyMC2. In realtà prima di poterlo sostituire definitivamente, OM necessita di alcune modifiche ed estensioni che permettano di garantire la generazione di risultati corretti per dataset arbitrariamente grandi. Tra tali modifiche certamente la più importante è l'estensione out-of-core, allo stato attuale soltanto parzialmente implementata, necessaria al fine di poter processare con successo dataset di decine di milioni di punti: come evidenziato anche dai test e dalle loro analisi riportati nel capitolo precedente, l'effettiva occupazione di memoria allo stato attuale risulta eccessiva già per dataset contenenti più di 10M di punti. Tale estensione è di certo affiancabile, al fine di limitare l'occupazione di memoria, anche dall'utilizzo all'interno del framework di tecniche per la rappresentazione compatta delle principale informazioni di interesse, come almeno punti e normali.

Un'altra possibile estensione cui fino ad ora non abbiamo ancora accennato ma che costituirebbe senza dubbi un ulteriore punto di forza di OM sarebbe quella per l'effettiva gestione anche del colore, in modo tale che la sua specifica risultasse integrata con il processo di ricostruzione della superficie. Al lato pratico questa estensione comporterebbe che per ogni vertice

non solo vengano determinate le coordinate spaziali ma siano anche specificate quelle di *texture*. Del resto qualora lo scanner disponga di un sensore a lui solidale per l'acquisizione del colore, le coordinate di texture risultano già associate ad ogni campione presente nelle range map; se invece tali informazioni non fossero generate già dalle apparecchiature di acquisizione, esse possono ancora essere ricavate a partire da fotografie ad alta risoluzione effettuate con macchine fotografiche digitali: in quest'ultimo caso le coordinate di texture da associare ad ogni vertice possono essere generate a partire dalla determinazione della trasformazione che permetta di allineare il colore con la geometria.

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] Pankaj K. Agarwal. Geometric range searching. In *CRC Handbook of Computational Geometry*. CRC, 1997.

[2] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[3] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421, New York, NY, USA, 1998. ACM Press.

[4] Nina Amenta and Yong J Kil. The domain of a point set surfaces. *Eurographics Symposium on Point-based Graphics*, 1(1):139–147, 6 2004.

[5] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. *ACM Trans. Graph.*, 23(3):264–270, 2004.

[6] Edward Angel. *Interactive computer graphics: a top-down approach with OpenGL*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[7] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Computational Methods in Applied Mechanics and Engineering*, 139:3–47, 1996.

[8] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cludio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[9] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[10] Lisa Gottesfeld Brown. A survey of image registration techniques. *ACM Comput. Surv.*, 24(4):325–376, 1992.

[11] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell. Smooth surface reconstruction from noisy range data. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 119–ff, New York, NY, USA, 2003. ACM Press.

[12] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, W. Richard Fright, Bruce C. McCallum, and Tim R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 67–76. ACM Press / ACM SIGGRAPH, 2001.

[13] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[14] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shape. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[15] Bernardini F. and Rushmeier H. The 3d model acquisition pipeline. *Computer Graphics Forum*, 21(2):149–172, 2002.

[16] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, 1992. ACM Press.

[17] Scott Schaefer Joe. Dual contouring: The secret sauce.

[18] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Siggraph 2002, Computer Graphics Proceedings*, pages 339–346. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2002.

[19] Leif Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In Eugene Fiume, editor, *Computer Graphics (SIGGRAPH-2001): Conference Proceedings*, pages 57–66, Los Angeles, USA, 2001. ACM.

[20] David Levin. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, pages 37–49, 2003.

[21] Thomas Lewiner, Hlio Lopes, Antnio Wilson Vieira, and Geovan Tavares. Efficient implementation of Marching Cubes cases with topological guarantees. *Journal of Graphics Tools*, 8(2):1–15, 2003.

[22] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[23] Jiri Matousek. Geometric range searching. *ACM Computing Surveys*, 26(4):421–461, 1994.

[24] Robert Mencl and Heinrich Mller. Interpolation and approximation of surfaces from three-dimensional scattered data points. In *DAGSTUHL '97: Proceedings of the Conference on Scientific Visualization*, page 223, Washington, DC, USA, 1997. IEEE Computer Society.

[25] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[26] Valerio Pascucci. Multi-resolution indexing for hierarchical out-of-core traversal of rectilinear grids, 2000.

[27] Valerio Pascucci and Randall J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 2–2, New York, NY, USA, 2001. ACM Press.

[28] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[29] Kari Pulli. Multiview registration for large data sets. In *3DIM*, pages 160–168. IEEE Computer Society, 1999.

[30] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Proceedings of the Third Intl. Conf. on 3D Digital Imaging and Modeling*, pages 145–152, 2001.

[31] Scott Schaefer and Joe D. Warren. Dual marching cubes: Primal contouring of dual grids. In *Pacific Conference on Computer Graphics and Applications*, pages 70–76. IEEE Computer Society, 2004.

[32] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004*. ACM Press, aug 2004.

[33] I. Söderkvist. Introductory overview of surface reconstruction methods. Technical Report 1999-10, Department of Mathematics, Lulea University of Technology, Lulea, Sweden., 1999. Can be retrieved from http://www.sm.luth.se/∼inge/publications/surfrec.ps.

[34] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[35] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, New York, NY, USA, 1994. ACM Press.

[36] Mason Woo, Davis, and Mary Beth Sheridan. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.